



**UNIVERSITÀ DEGLI STUDI DI ROMA  
TOR VERGATA**

MACROAREA DI SCIENZE MATEMATICHE, FISICHE E  
NATURALI

CORSO DI LAUREA IN INFORMATICA

A.A.

**Tesi di Laurea**

Blockcerts: un possibile standard per la verifica autonoma  
delle certificazioni

**RELATORE**

Prof. Francesco Pasquale

**CANDIDATO**

Marco Nardi



# Indice

<b>Introduzione</b>	<b>1</b>
<b>Organizzazione della Tesi</b>	<b>3</b>
<b>1 Introduzione alla tecnologia di Bitcoin</b>	<b>5</b>
1.1 Il Creatore: Satoshi Nakamoto . . . . .	5
1.2 Cos' è Bitcoin? Una breve introduzione . . . . .	7
1.2.1 L'Hashing . . . . .	7
1.2.2 Signature . . . . .	10
1.2.3 Public Key e Secret Key . . . . .	12
1.3 Transazioni nel Network . . . . .	13
1.3.1 Modello UTXO . . . . .	14
1.3.2 Struttura di una Transazione . . . . .	15
1.3.3 Fee delle Transazioni . . . . .	18
1.3.4 Transaction Mining Process . . . . .	19
1.3.5 Il Block Header . . . . .	20
1.3.6 Ordine delle Transazioni all'interno di un Blocco . . . . .	23
1.4 Il Mining . . . . .	23
1.5 Decentralizzazione di Bitcoin . . . . .	25
1.5.1 Tendenza verso la Centralizzazione . . . . .	26

<b>2</b>	<b>Blockchain: il public ledger di Bitcoin</b>	<b>29</b>
2.1	La Catena di Bitcoin . . . . .	30
2.2	Accenno alle Fork . . . . .	32
2.2.1	Soft Fork . . . . .	33
2.2.2	Hard Fork . . . . .	34
2.2.3	Esempi di Fork . . . . .	35
2.3	La Proof of Work . . . . .	36
2.3.1	Il Mining e la Proof of Work . . . . .	37
2.4	Difficulty Adjustment Algorithm . . . . .	38
<b>3</b>	<b>Introduzione a Blockcerts</b>	<b>41</b>
3.1	Il progetto Blockcerts . . . . .	42
3.2	Esempio di caso d'uso . . . . .	44
3.3	Il processo di Blockcerts . . . . .	46
3.4	Creazione, emissione e visione di certificati locali . . . . .	49
3.4.1	Creazione di un'immagine su Docker . . . . .	50
3.4.2	Libreria cert-issuer . . . . .	53
<b>4</b>	<b>Personalizzazione dei certificati: la libreria cert-tools</b>	<b>58</b>
4.1	Configurazione dell'issuer ID . . . . .	60
4.2	Creazione dei certificati non firmati . . . . .	63
4.2.1	Creazione del template . . . . .	63
4.2.2	Creazione delle istanze . . . . .	63
4.3	Firma dei certificati all'interno della testnet di Bitcoin . . . . .	64
4.4	Visualizzazione del Certificato tramite Blockcerts . . . . .	67
4.5	Visualizzazione app Android . . . . .	69

<b>5</b>	<b>Un ambiente user-friendly per Blockcerts</b>	<b>70</b>
5.1	Le problematiche incontrate . . . . .	71
5.2	Un approccio laborioso . . . . .	72
5.3	Gestione del processo tramite Python . . . . .	72
5.3.1	Lato Organizzativo: gestione del file roster . . . . .	73
5.3.2	Lato Issuer: inserimento dati e firma dei certificati . . . . .	75
<b>6</b>	<b>Conclusioni</b>	<b>80</b>
	<b>Elenco delle figure</b>	<b>82</b>

# Introduzione

In questi ultimi anni si è parlato molto delle criptovalute, della tecnologia Blockchain e di come potessero rivoluzionare il nostro mondo. Da ciò si sono generati tantissimi progetti ed idee, tra cui quello di utilizzare la tecnologia delle blockchain per poter scrivere delle informazioni all'interno di essa, che possano persistere nel tempo in modo sicuro, senza poter essere modificate o falsificate.

Il progetto Blockcerts[3] prende questa idea e la applica al mondo delle certificazioni, ovvero ha sviluppato un modo per poter emettere, all'interno della blockchain stessa, un insieme di certificati. Questi certificati, una volta emessi, non potranno essere modificati, falsificati o cancellati dalla blockchain stessa, di conseguenza saranno persistenti e visibili da tutti coloro che ne hanno accesso. Inoltre, il beneficiario sarà in possesso delle credenziali necessarie alla sua verifica.

Attualmente, Tor Vergata non dispone di un servizio di digitalizzazione dei certificati di Laurea su blockchain, ho perciò pensato di utilizzare Blockcerts come una base per proporre la struttura di un possibile servizio. Il progetto e le sue librerie sono disponibili all'interno di un repository online ma sono indirizzate ad un pubblico di sviluppatori, rendendo l'adozione del progetto difficile per gli enti che si vogliono approcciare ad esso.

Tramite il mio lavoro, esplorerò i vari processi dietro queste librerie mostrando come poter creare e firmare dei certificati personalizzati, sia tramite Docker[6], sia tramite

la propria macchina con l'ausilio di Bitcoin Core[2]. Inoltre, presenterò una mia interfaccia GUI, in Python, che permetta di approcciarsi a queste librerie in un modo più user-friendly e senza dover avere delle conoscenze del progetto stesso o di Bitcoin, con il fine di rendere un primo approccio al progetto più semplice per gli enti che si affacciano ad esso.

La digitalizzazione dei certificati è uno dei passi per avvicinarsi ad un mondo sempre più tecnologico e decentralizzato. Il mio progetto è ancora all'inizio del suo sviluppo ma permetterebbe un avvio graduale verso la digitalizzazione dei certificati di Laurea dell'università di Tor Vergata. Infatti esso può essere utilizzato in parallelo con l'attuale sistema di emissione dei certificati cartacei, senza dover effettuare delle modifiche allo stesso o istruire gli utenti al suo utilizzo.

# Organizzazione della Tesi

Nel Capitolo 1 introdurremo nel dettaglio le tecnologie che caratterizzano bitcoin, cercando di spiegare come è strutturato bitcoin nella sua interezza. Parleremo anche delle transazioni e degli indirizzi bitcoin.

Nel Capitolo 2 parleremo della Blockchain, ledger pubblico di bitcoin che rende possibile la decentralizzazione della moneta. Questa tecnologia è alla base di Bitcoin e di Blockcerts ed è l'argomento base di questa tesi. Oltre ad essa e alla sua struttura, parleremo della Proof of Work e del mining, argomenti interconnessi tra loro e che hanno reso possibile la costruzione della catena di Bitcoin fino ad oggi.

Nel Capitolo 3 parleremo, infine, dell'argomento principale della tesi, il progetto Blockcerts. Dopo averlo introdotto ed aver esposto alcuni casi d'uso, mostreremo come firmare dei certificati campioni, forniti dalle librerie utilizzate, all'interno di una regtest di bitcoin e mostreremo come replicare il procedimento tramite l'utilizzo del software Docker.

Nel Capitolo 4 mostreremo come personalizzare questi certificati modificando le informazioni personali dell'emittente, del beneficiario e i loghi. Affronteremo, passo passo, la creazione del certificato personalizzato, discutendo su come esso possa essere modificato. Mostreremo poi come è possibile visualizzare e verificare i certificati appena creati tramite l'ausilio del sito web e dell'applicazione android di blockcerts.



Infine, nel Capitolo 5 mostrerò un'applicazione creata da me in Python, tramite il supporto della libreria QT, che aiuta a semplificare il processo di creazione di un certificato, rendendo il tutto il più user friendly possibile.

# Capitolo 1

## Introduzione alla tecnologia di Bitcoin

In questo capitolo introduttivo ci accingiamo a presentare Bitcoin, la famosa criptovaluta che è diventata l'icona delle monete decentralizzate. Ci soffermeremo brevemente sulla sua storia, sul suo funzionamento e, soprattutto, sulla sua tecnologia. Affronteremo tutti i gli aspetti di questa moneta, partendo dal semplice concetto di hash per arrivare, infine, al concetto più complicato di Proof of Work. In questo percorso parleremo anche della Blockchain, colonna portante di Bitcoin, e del concetto di proof of work, ingranaggio del sistema di Bitcoin. Se si ha già una conoscenza approfondita di Bitcoin e delle sue tecnologie, consiglio la lettura dal Capitolo 3 dato che, il Capitolo 1 punta a dare una solida base a chiunque si stia avvicinando, per la prima volta, al mondo di Bitcoin e della permissionless blockchain.

### 1.1 Il Creatore: Satoshi Nakamoto

Il dominio “bitcoin.org” fu registrato il 18 agosto 2008 tramite il sito “anonymous-speech.com” e, il primo novembre dello stesso anno, in una mail list di Criptografia venne postato un link che portava ad un White Paper intitolato “Bitcoin: A Peer-to-Peer Electronic Cash System” il cui autore era Satoshi Nakamoto. Il 3 gennaio

2009 il network di bitcoin prese vita, sempre grazie a Satoshi che “minò” il primo blocco in assoluto chiamato “Genesis Block of Bitcoin”, ovvero il blocco numero zero dell’attuale Blockchain. All’interno di questo blocco, con un reward di 50BTC, troviamo la frase:

“The Times 03/Jan/2009 Chancellor of brink of second bailout for banks”

Testo facente riferimento a una testata giornalistica britannica di Londra, il “Times” pubblicata proprio in quel giorno. Nonostante questo precedente, il primo software open source, versione 0.1 del software di Bitcoin, fu rilasciato al pubblico solo sei giorni dopo, il 9 gennaio 2019, tramite l’host “SourceForge”. Questa milestone fu la prima di una lunga storia di transazioni del network di Bitcoin che lo hanno trasformato da un semplice progetto Open Source ad una delle più famose criptovalute in esistenza. Presenta un volume di scambi giornaliero pari ad un giro di affari di €90.647.817.557,00, con il valore di 1 BTC pari a €46.552,80, nella giornata del 21/02/2021. [5]

Nakamoto scomparve dalla scena pubblica il 12 dicembre 2010 con un ultimo post sul sito “bitcointalk.org”. Ad oggi non si conosce la sua reale identità, né se si tratta davvero di un’unica persona. Si possono fare solo speculazioni su di essa, sul se fosse o meno il nickname di un’organizzazione o di un gruppo di persone, sulla sua nazionalità ecc.

La cosa certa è che Nakamoto, durante la crescita di Bitcoin e del suo lavoro, è stato l’ideatore del primo database in stile blockchain e, nel processo, risolse anche il problema tipico delle monete digitali, il double-spending. [4]

## 1.2 Cos' è Bitcoin? Una breve introduzione

Bitcoin è una criptovaluta completamente decentralizzata, ovvero non ha una banca centrale che emette o gestisce la valuta stessa. Di conseguenza, questa valuta può essere inviata, da User ad User, senza veri e propri intermediari e tramite un network peer-to-peer. Le uniche figure presenti all'interno del sistema che si avvicinano alla figura di intermediari sono i cosiddetti “Miners”, fulcro del network stesso; essi si occupano di verificare le transazioni generate tramite questo scambio e di registrarle all'interno di un registro pubblico, ovvero la Blockchain.

### 1.2.1 L'Hashing

Prima di parlare della tecnologia di bitcoin, dobbiamo chiarire dei concetti, fondamentali della tecnologia stessa di bitcoin, come quello delle funzioni di crittografiche di hashing. Una funzione Hash Criptografica è un algoritmo matematico che trasforma una mole di dati, detta messaggio, di dimensione arbitraria in una stringa di numeri di dimensione fissa apparentemente casuale. Apparentemente perché, in realtà, è una funzione deterministica, ovvero se diamo di nuovo in pasto lo stesso input alla funzione, riottenremo sempre lo stesso output.

$$\text{hash}(\text{dati}) = \text{Output}$$

Dati può essere di una qualsiasi grandezza mentre Output sarà sempre della stessa grandezza. Dei dati codificati tramite una funzione di hash sono soggetti all'effetto “avalanche”, ovvero cambiare anche solo 1 bit nell'input cambierebbe, all'incirca, metà dei bit in Output. Questo rende i messaggi non modificabili, a meno che non si modifichi anche l'output.

Una funzione crittografica hash è definita “well defined” se rispetta le proprietà di *Preimage Resistance*, *2nd Preimage Resistance* e *Collision Resistance*.

- *Preimage Resistance* è una proprietà estremamente importante; afferma che, data una qualsiasi  $y$ , non possiamo trovare in nessun modo banale una  $x$  tale che  $hash(x) == y$ . Con il termine *Preimage* intendiamo i dati che avevamo prima dell'output, una buona funzione hash crittografica deve essere in grado di evitare qualsiasi tentativo banale per poter risalire ai dati utilizzati in input dato un qualsiasi output. Ovviamente si può risalire ai dati precedenti tramite una serie di try-error dove si prova un dato e si controlla che l'output corrisponda. Questa tecnica funziona ma il numero di try-error richiesti è estremamente grande e si aggira sui  $2^{256}$  nel caso di Bitcoin<sup>1</sup>, cioè  $10^{78}$  possibilità, cosa che lo rende impossibile da calcolare tramite un approccio Brute Force.
- Mentre la *2nd Preimage Resistance* descrive come, se abbiamo una  $x$  e una  $y$  tale che  $hash(x) == y$ , non è possibile trovare banalmente una  $x'$  tale che  $x' \neq x \wedge hash(x') == y$ .
- *Collision Resistance* definisce come nessuno possa trovare una  $x$  e una  $z$  tale che  $x \neq z \wedge hash(x) == hash(z)$ . Almeno non con modi banali, infatti è possibile trovarla tramite un approccio di Brute Force, ma il tempo necessario è sempre di  $2^{256}$  operazioni. Con una buona implementazione, e logica, è possibile applicare un tipo di attacco di Brute Force; questo tipo di attacco

---

<sup>1</sup>L'algoritmo utilizzato da Bitcoin, per le funzioni di Hash, è lo SHA-256 [13]. Esso prende in Input un dato di grandezza arbitraria e restituisce, in Output, 256 bit. Ogni bit può assumere il valore 0 o 1, di conseguenza abbiamo  $2^{256}$  combinazioni differenti.

si chiama “Birthday Attack”<sup>2</sup> e si basa sul semplice fatto che, ogni prova che facciamo, ci dà un output. Possiamo usare questa informazione, integrandola nelle nostre operazioni di brute force, per cercare di ridurre il numero di tentativi. Questo stile di attacco permette di abbassare il numero di tentativi a  $2^{128}$ , numero ancora molto alto per un semplice attacco di brute force.

Alcune delle funzioni hash più famose, come SHA-1 e MD5, sono state analizzate a fondo e state dimostrate “broken”. Questo termine sta ad indicare che, l’input di una determinata funzione Hash può essere trovato in un tempo gestibile tramite delle operazioni logiche e degli aggiustamenti all’algoritmo; un Team di ricerca di Google è riuscito a trovare due valori, usando molto potere computazionale e tempo,  $x$  e  $z$  che, sottoposti a quella determinata funzione hash, davano lo stesso risultato. [15]

Questo risultato basta per far deprecare l’utilizzo di una funzione hash crittografata, nel nostro caso di SHA-1 e MD5, perché “rompe” la proprietà Collision Resistance di conseguenza non è sicuro utilizzarla. D’altro canto, non ci sono Preimage Attack per queste funzioni, quindi dato un hash output, non è possibile trovare l’input. [14]

---

<sup>2</sup>Chiamato così per indicare il Birthday Paradox nel campo della probabilità. Esso afferma che, in una stanza con 23 persone o più, la probabilità che due persone condividano lo stesso compleanno sono maggiori del 50%. [12]

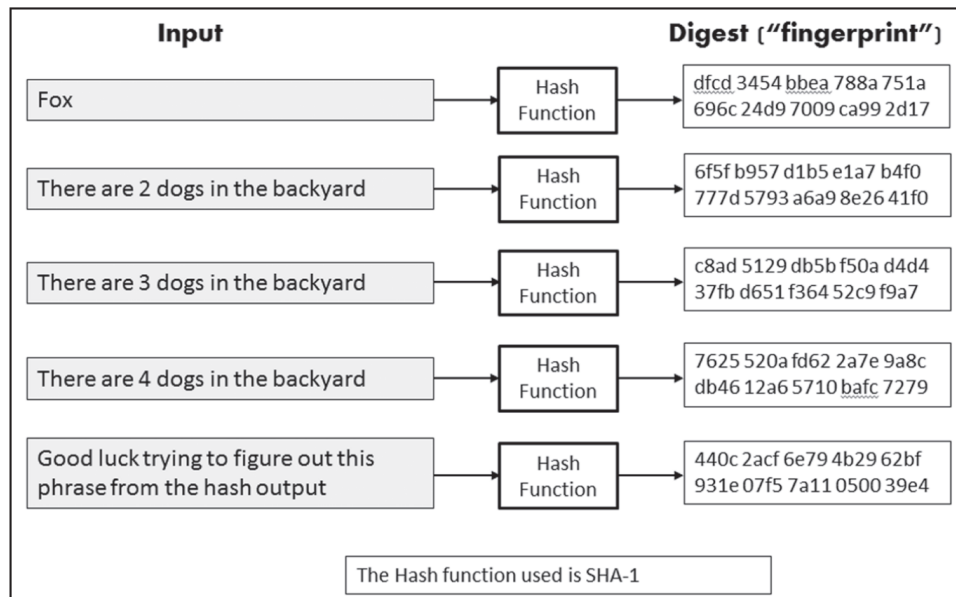


Figura 1.1: Esempi di Input e Output di una funzione hash criptografica

Possiamo notare come, nella figura 1.1, l'hash della parola "Fox" e della frase "There are 2 dogs in the backyard" siano estremamente differenti tra loro e non riportino nessun indizio che possa associarli alla preimage corrispondente. Inoltre, possiamo notare come, nella frase "There are 2 dogs in the backyard" e "There are 3 dogs in the backyard" cambi solo un singolo bit ma, nella traduzione in hash tramite SHA-1 l'output ottenuto sia estremamente differente. Come abbiamo descritto prima, questo è esattamente l'effetto Avalanche, ovvero basta cambiare un singolo bit nella preimage per avere un output differente dal precedente di, almeno, la metà dei byte. [14]

### 1.2.2 Signature

Ora che abbiamo definito il concetto di hash, possiamo utilizzarlo per definire un elemento chiave di bitcoin. Una Signature all'interno dell'ecosistema di Bitcoin può essere vista come una firma, applicata ad un messaggio, che attesta la nostra identi-

tà. È un elemento estremamente importante nella struttura di Bitcoin che permette di verificare, con l'ausilio della funzione *Verify*, l'identità dell'User che, per esempio, inizializza una transazione. Per definire meglio una Signature, spiegherò l'utilizzo di *GenerateKey*, *Sign* e *Verify*, funzioni pilastro le cui azioni sono necessarie per creare un qualsiasi schema di Signature.

- Tramite la funzione *GenerateKey*, la quale non prende in Input nessun argomento, generiamo una coppia di chiavi. Una delle chiavi generate sarà la Key Pubblica, che mostreremo al mondo e che designeremo come nostra “identità” e una Key segreta che dovrà essere utilizzata per provare l'identità dello User, ovvero provare che dei messaggi siano effettivamente firmati da quell'identità. Bisogna notare che non c'è nessun collegamento tra una Public Key e la vera identità di un soggetto, difatti una persona può possedere più Public Key ed usarle senza problemi, come più persone potrebbero, potenzialmente, condividere la stessa Public Key. La Secret Key non dovrà essere resa pubblica, pena la compromissione della coppia di Key dato che, chiunque entri in possesso di essa potrà spacciarsi per lo User principale all'interno del Network.
- La funzione *Sign* prende in Input una Secret Key e un Messaggio. Tramite di essa, colui che detiene una Secret Key può firmare un determinato messaggio. Così facendo, il messaggio verrà fatto passare all'interno di una funzione di hashing insieme alla Secret Key, legandoli insieme, e restituendo un output completamente differente.
- Tramite la funzione di *Verify*, che prende in input una Public Key, un Messaggio e una Signature, chiunque possenga la Public Key in questione potrà verificare la coppia Messaggio-Signature, ovvero verificare che il messaggio in que-



stione sia stato codificato, con una funzione Sign, tramite l'utilizzo della Secret Key corrispondente alla Public Key utilizzata. Questa funzione restituisce un boolean a seconda se corrispondono o meno. [14]

### 1.2.3 Public Key e Secret Key

Bitcoin, per il suo network, utilizza un sistema crittografato chiamato “Public Key Cryptography”, ovvero un sistema Asimmetrico di Encrypting che utilizza una coppia di chiavi, chiamate Secret Key e Public Key, per firmare e verificare dei messaggi. Tutti i BTC sono registrati su degli indirizzi specifici di Bitcoin. Questi indirizzi sono facili da creare, come abbiamo visto nel 1.2.2 a pagina 10, infatti non richiedono altro che creare una Secret Key in modo casuale e calcolare la corrispondente Public Key. Questa operazione però non è reversibile, infatti non è possibile partire da una Public Key e forgiare la corrispondente Secret Key, infatti essa richiederebbe un tempo estremamente lungo tramite un approccio di Brute Force, così lungo da non risultare fattibile. Questa coppia di Key permette, ad un Utente Bitcoin di poter interagire con la rete e di poter creare delle transazioni in totale autonomia. Lo User deve solo utilizzare la propria Secret Key per firmare la transazione emessa, in modo tale che, la rete Bitcoin possa verificare la firma tramite la Public Key. [7]

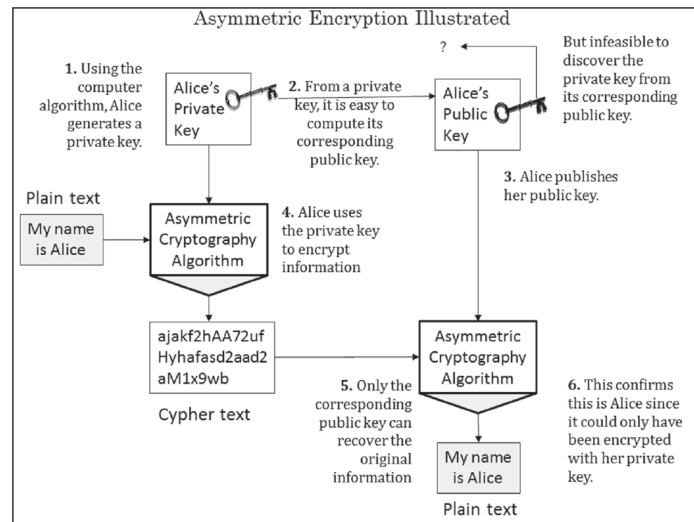


Figura 1.2: Illustrazione Logica dell'Algoritmo di Criptografia Asimmetrico

In ogni singolo processo di Bitcoin, la Secret Key non è mai rivelata, infatti se essa fosse condivisa con il web, l'indirizzo di Bitcoin pubblico associato sarebbe da considerare compromesso. Stessa cosa vale se l'utente perde il possesso della Secret Key del suo indirizzo, poiché ciò renderebbe ogni BTC assegnato ad esso completamente inutilizzabile dato che non non la si potrebbe recuperare se non tramite Brute Force ma, come abbiamo già discusso in precedenza, è un'eventualità impossibile. Tramite questo processo è tecnicamente possibile creare una Secret Key già esistente, ma le probabilità che ciò possa accadere sono infinitamente piccole. Di conseguenza è pressoché improbabile che un Address Bitcoin venga compromesso per questo motivo; ovvero che, durante una creazione di una coppia di Key, venga creata una Key già esistente. [14]

## 1.3 Transazioni nel Network

All'interno dell'ecosistema di Bitcoin, possiamo vedere una transazione come un trasferimento di denaro, token BTC, da un indirizzo *A* ad un indirizzo *B*. A gran-

di linee possiamo dire che quest'azione avviene tramite la creazione di un messaggio dove viene specificato l'indirizzo del destinatario (Public Key) e l'importo, in satoshi, da inviare. Questo messaggio viene poi firmato dal mittente, tramite la Secret Key, e spedito a tutti i componenti della rete peer-to-peer (vedi 1.2.2 pagina 10). La transazione, prima di essere effettiva, deve essere "approvata" tramite un procedimento specifico, che verrà approfondito più avanti quando introdurremo il concetto di Blockchain e di Mining. In ogni caso, dopo che la transazione viene gestita e il destinatario riceve i BTC pattuiti, essi potranno essere spesi, anche immediatamente, tramite la creazione di una nuova transazione.

### 1.3.1 Modello UTXO

Bitcoin utilizza un modello chiamato UTXO (*Unspended Transaction Output*) per la gestione delle transazioni; esso si contrappone all'Account Based Model<sup>3</sup> classico utilizzato da Ethereum e dalle Banche che siamo abituati a conoscere. Il modello ABM, infatti, tiene una lista di account e di bilanci all'interno di un database e, ogni volta che avviene una transazione, essa viene dichiarata valida soltanto se il bilancio di quell'account la permette. (In questo modello il Sender viene visto come un debitore e il Receiver come un accreditato)

Il modello UTXO di Bitcoin invece è estremamente differente dal classico sistema bancario; in questo modello ogni singolo coin è univoco e, durante una transazione, si fa riferimento ad uno specifico coin da spendere. Al momento della transazione, il coin che si vuole spendere viene consumato e, al suo posto, ne viene generato uno completamente nuovo che verrà assegnato all'indirizzo Bitcoin di destinazione al posto del precedente coin. Ogni coin generato potrà essere speso solo una volta al-

---

<sup>3</sup>Modello che si basa sul concetto di conto o saldo. Di seguito un link per l'approfondimento generale dell'argomento: <https://academy.horizen.io/technology/expert/utxo-vs-account-model/>

l'interno della Blockchain, cosa che permette di mitigare il problema, presente nel modello ABM, del "Replay Attack" <sup>4</sup>.

Questo sistema permette anche di spendere una singola parte di coin; infatti se si vorrà spendere solo una singola moneta, la transazione, che si verrà a creare in quel momento, distruggerà l'intero coin e, successivamente, ne creerà uno del valore che si vuole spedire al destinatario, insieme ad un altro del valore restante, che verrà spedito indietro al mittente. [14] Lezione 4

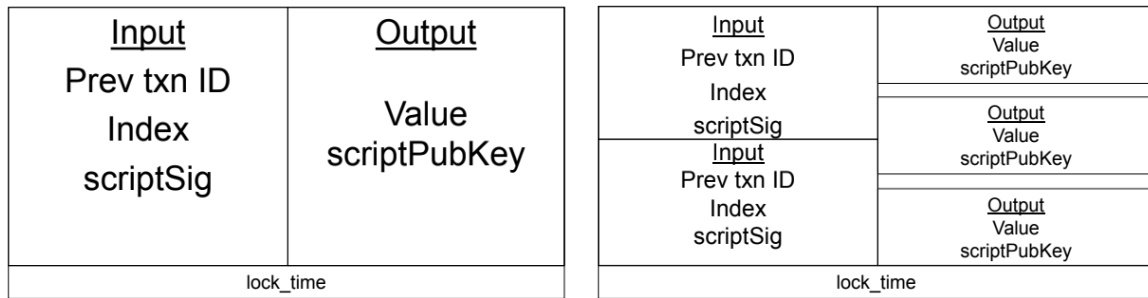
### 1.3.2 Struttura di una Transazione

Precedentemente abbiamo definito grossolanamente la transazione in Bitcoin, ma in realtà ha un funzionamento più complesso. La rete Bitcoin non tiene conto di uno specifico balance collegato ad un account o simili, di conseguenza quando vogliamo spendere dei coin, dobbiamo fare riferimento a delle transazioni passate, da cui abbiamo ricevuto delle monete, e spenderli tramite di esse.

Possiamo quindi suddividere le transazioni in Input( le transazioni da cui prendiamo i soldi) e in Output (il valore che spediremo e l'indirizzo che lo riceverà). Nel caso in cui una singola transazione non basti a coprire lo spostamento di coin, potremmo fare riferimento a più transazioni passate in contemporanea. Così facendo avremmo più transazioni in Input nella stessa transazione. Ma una transazione può anche avere più output. Possiamo quindi affermare che una transazione può avere  $N$  input e  $M$  output. [14]

---

<sup>4</sup>È un attacco che utilizza un "replay" dei dati trasmessi da un Sender differente



(a) Struttura con Singolo I/O

(b) Struttura con Multitpli I/O

Figura 1.3: Struttura Logica Interna di una Transazione

Come possiamo osservare in 1.3, in Input abbiamo un valore “*Prev txn ID*” che fa riferimento ad una transazione precedente, tramite il suo Hash, dalla quale vogliamo spendere l’Output, un “*Index*” che indica quale output della transazione precedente, indicata da *Prev txn ID*, si vuole spendere e la “*scriptSig*”, che non è altro che l’autorizzazione per spendere quel determinato output (ovvero la Signature del mittente della transazione).

Infine, il parametro “*lock time*” fa riferimento a quando, questa transazione nella sua interezza, possa essere inserita all’interno di un blocco; il valore di questa variabile può essere espresso sia come istanza temporale, sia come altezza in blocchi. Le variabili *Prev txn ID* e *Index* appena descritte possono, da sole, identificare univocamente un Output di una qualsiasi transazione all’interno della blockchain.

Consumando l’output precedente, creiamo un nuovo output con un valore “Value” di coin, identificato da un valore rappresentante il numero di satoshi che si vogliono spendere di quell’output, che verrà assegnato ad una Public Key, di possesso del destinatario, tramite la funzione *scriptPubKey*.

Nella transazione, *scriptPubKey* è una funzione che specifica le condizioni per cui quell’output può essere riscattato. Un esempio di caso d’uso in Bitcoin che quell’output può essere riscattato da chiunque possa produrre una Signature, della corri-

spondente Secret Key, con quella Public Key.

Nelle transazioni con più input e/o più output, vedi Figura 1.3b, ogni Input avrà la propria *scriptSig* mentre ogni output avrà la propria *scriptPubKey*. Questo avviene perchè si possono attingere coin da più output precedenti, anche se assegnati a differenti Public Key. Al contrario, gli output di una transazione non devono per forza condividere la Pub Key a cui vengono assegnati, richiedendo così una prova differente per ognuno di loro.

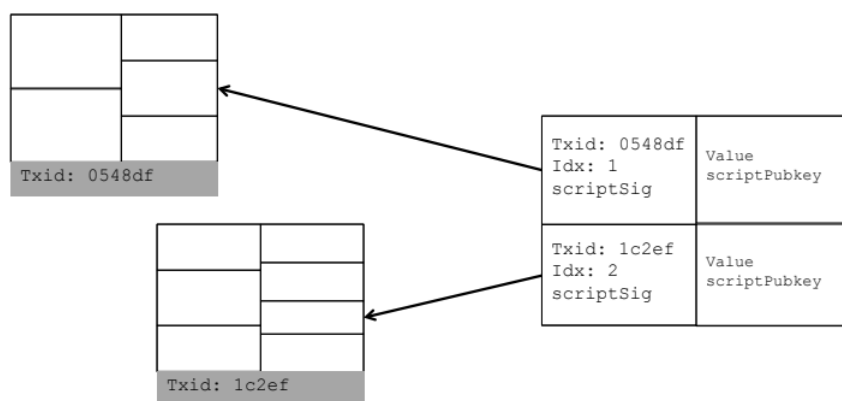


Figura 1.4: Struttura Logica di una Transazione

Dato che ogni coin viene “bruciato” quando viene spostato, non è possibile utilizzare due volte lo stesso output. Infatti, la rete Bitcoin prenderà nota del fatto che, un determinato output, è stato speso e dichiarerà non valida qualsiasi transazione che proverà a spende un output già utilizzato. Eviterà così che i coin, di un qualsiasi output nella blockchain, vengano spesi più volte<sup>5</sup>. [14]

<sup>5</sup>Quando utilizziamo delle monete virtuali stiamo scambiando dei dati, dobbiamo perciò essere sicuri che quei dati non vengano copiati e utilizzati più volte. Questo è il problema del Double Spend.

### 1.3.3 Fee delle Transazioni

Anche nella rete Bitcoin le transazioni richiedono delle fee per “sostenere” la convalida, anche se queste fee non sono obbligatorie e sono a discrezione dello User. Esse non devono essere specificate all’interno delle transazioni, infatti vengono implicitamente pagate nel momento in cui non si spende, tramite i vari output, tutti i BTC messi in input. In altre parole, le fee sono implicate nella transazione, ogni Satoshi che non viene speso al suo interno viene implicitamente trattato come fee della transazione, fee che vengono poi “collezionate” dai Miner (argomento che verrà trattato nella sezione 1.4). Nelle transazioni di Bitcoin le fee sono opzionali ma “sperano” i Miner a scegliere quella transazione per processarla, in cambio di elettricità, con più o meno priorità a seconda del valore della fee. Transazioni con fee basse, o inesistenti, potrebbero rimanere invalidate per moltissimo tempo, continuamente sorpassate da transazioni con fee maggiori<sup>6</sup>.

A governare il mercato delle fee non sono che gli User, di conseguenza le fee vengono pagate in base a quanto sia disposto a pagare lo user rispetto al suo concetto di priorità. Inoltre, se si analizzano le fee delle transazioni, insieme al loro tempo di convalida, possiamo notare che, spesso, chi ha pagato delle fee alte per avere una priorità altrettanto alta, si è visto convalidare la transazione insieme a di chi ha pagato fee basse per avere una priorità altrettanto bassa nelle giornate in cui le transazioni erano poche e non gravavano sulla rete. [14]

---

<sup>6</sup>Se il Network fosse estremamente congestionato, delle transazioni che non hanno delle fee implicate non verrebbero processate dato che, ogni transazione successiva avrà più priorità. Questo problema può essere rapportato, con un pò di fantasia, al problema dello Starvation dei sistemi operativi.

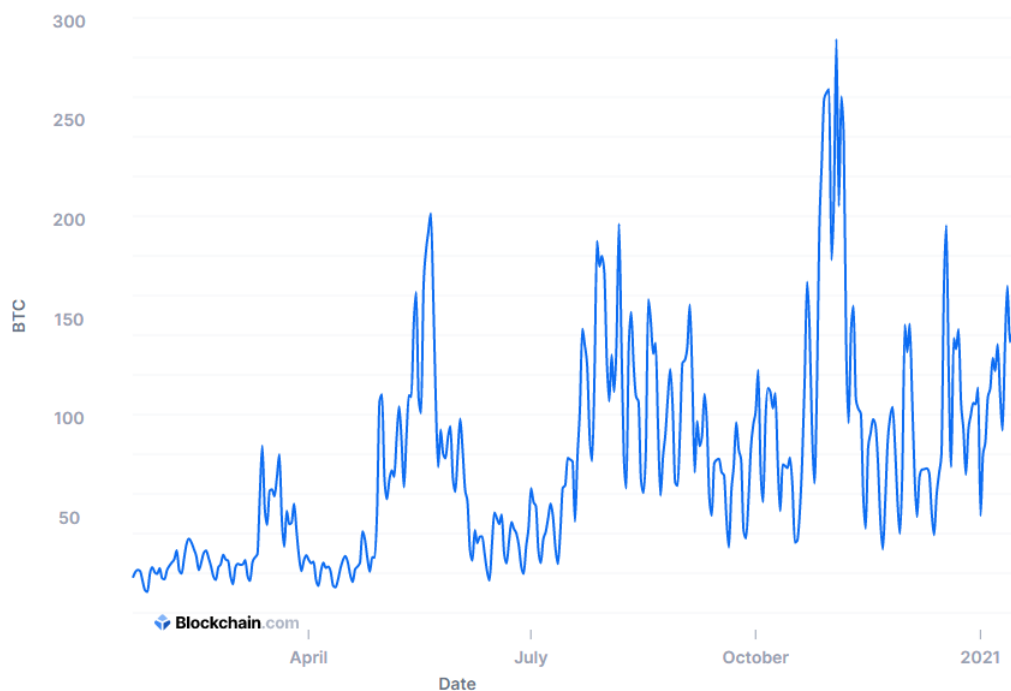


Figura 1.5: Grafico del totale delle fee, in BTC, pagate ai Miner dal 2020 al 2021

Come possiamo vedere dalla figura, le fee medie nel circuito Bitcoin cambiano in continuazione; l'argomento delle fee è estremamente complesso e non verrà approfondito all'interno di questa tesi.

### 1.3.4 Transaction Mining Process

Abbiamo affrontato il discorso delle transazioni, la loro struttura e delle fee; ma come vengono gestite queste transazioni nel network? Cerchiamo di rispondere a questa domanda introducendo il ruolo dei *Miner*, che approfondiremo nel capitolo successivo, e il concetto di *blocco*.

Gli utenti di Bitcoin che vogliono effettuare una transazione, dopo averla creata, la trasmettono in broadcast all'interno del network di bitcoin; da lì dei nodi peer-to-peer di Bitcoin, chiamati Miner, prendono tutte le transazioni che ricevono in



broadcast, le ritrasmettono se necessario, e le impacchettano all'interno di un blocco. Dopo aver costruito un blocco ed effettuato un "lavoro", queste transazioni vengono confermate e il blocco successivo, con le transazioni successive, viene costruito.

### 1.3.5 Il Block Header

Il blocco creato e convalidato è identificato da un "*Block Header*", esso è utilizzato per identificare un particolare blocco all'interno della Blockchain.

Approfondiremo nel Capitolo 2 il discorso della Blockchain, ma per ora accenniamo al fatto che consiste in un insieme di blocchi concatenati tra loro. Ogni blocco viene utilizzato per salvare delle transazioni che avvengono nel Network Bitcoin e ogni singolo blocco contiene un suo unico Header, di conseguenza potrà essere identificato inequivocabilmente tramite esso.

Possiamo vedere l'Header di un blocco come il messaggio che deve soddisfare la Proof of Work, vedi Capitolo 2.3 pagina 36; è composto da 80 byte e le sue componenti sono: "Version", "Prev Hash", "Merkle Root", "Time", "Diff" e "Nonce".

- *Version* indica la versione del software Bitcoin, campo da 4 byte utilizzato per dare indicazioni della versione del software che ha gestito il blocco in questione; utile durante delle fork, argomento approfondito nel Capitolo 2 pagina 32, anche se, attualmente, non è utilizzato in modo corretto e non ha molto peso.
- Il campo *Time* indica il tempo in secondi, dal 1970, dalla creazione del blocco. Esso è da 4 byte ed è utilizzato per capire con che frequenza vengono generati i blocchi nell'arco di un lasso di tempo. Questo ci è utile perchè il processo di creazione dei blocchi è controllato da un algoritmo specifico, approfondito nel Capitolo 2.4 a pagina 38, che ne determina la difficoltà di creazione in modo tale che, 2016 blocchi, vengano creati nell'arco di 14 giorni circa.

- Anche il campo *Diff* è da 4 byte ed è dedicato ad indicare la difficoltà di creazione del blocco. Il valore di questo campo può essere calcolato, quindi è considerato superfluo.
- Nonce è un campo libero, ci si può inserire di tutto ed è di grandezza 4 byte. Questo campo è estremamente sfruttato durante il Mining, come vedremo meglio nella Sezione 2.3.1 a pagina 37.
- *Prev Hash* fa riferimento all'hash del blocco precedente ed è di 32 byte. Grazie ad esso possiamo creare effettivamente la “catena di blocchi”, al contrario del campo *Nonce* che non è altro che un insieme di dati la cui utilità è creare del “lavoro”.
- Il *Merkle Root* anche è da 32 byte, esso è un valore Hash che rappresenta un albero binario formato dall'insieme delle transazioni il cui Hash viene concatenato. L'albero serve per poter facilmente dimostrare che una determinata transazioni faccia parte di un blocco. Questo, però, senza dare, effettivamente, l'intero blocco al richiedente.

L'idea dietro al Merkle Root è relativamente semplice. Prendiamo i TXID delle transazioni, che non sono altro che l'hash della transazione stessa, e li mettiamo come foglie dell'albero che stiamo costruendo, tramite un approccio bottom-top. Concatenando i valori delle foglie, che hanno lo stesso padre, tra loro e creando un hash del risultato, otteniamo un valore di hash che assegneremo al padre delle due foglie. Questo procedimento, viene ripetuto fino a raggiungere la radice dell'albero.

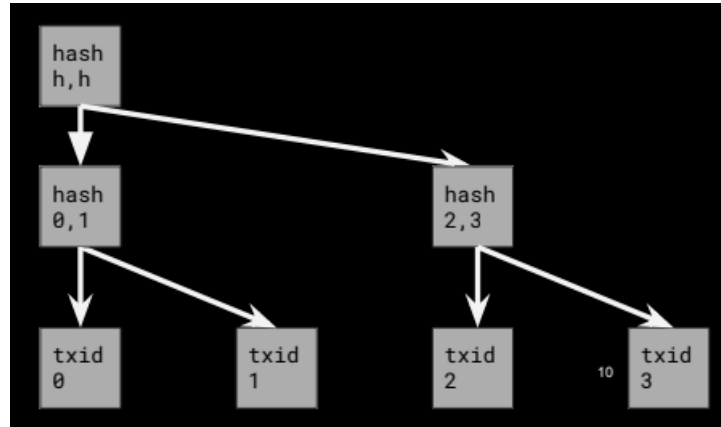


Figura 1.6: Esempio di un Merkle Root

Ipotizziamo che, il nostro compito nella rete sia quello di un Full Node<sup>7</sup>.

Se un nodo volesse controllare che all'interno di un blocco ci sia effettivamente una transazione, per esempio la *transazione 1* (txid1), ci basterà fornirgli solo l'hash TXID della *transazione 0*<sup>8</sup> e l'hash  $h(2, 3)$ . Il richiedente potrà calcolarsi da solo l'hash  $h(0, 1)$  e, con quello, calcolarsi l'hash  $h(h, h)$  concatenandolo con l'hash  $h(2, 3)$ .

Questo procedimento permette di risparmiare moltissimo spazio all'interno della rete dato che verranno inviati  $O(\log n)$  hash per un singolo check di un albero, invece di  $n$  hash nel caso in cui la Merkle Root si fosse calcolata concatenando tutti gli hash delle transazioni per poi effettuare l'hash del risultato. Possiamo quindi considerare la radice di questo albero come un hash combinato di tutte le transazioni all'interno del blocco.

È grazie all'Header che possiamo creare una “catena di blocchi” dato che, ogni blocco farà riferimento al suo blocco immediatamente precedente inserendone l'header, nel campo *Prev tx ID*. [14] Lezione 5

<sup>7</sup>Un Full Node è un programma che convalida totalmente delle transazioni e dei blocchi nella rete Bitcoin.

<sup>8</sup>La transazione 0 è la transazione “coinbase”

### 1.3.6 Ordine delle Transazioni all'interno di un Blocco

All'interno del Merkle Root, la Transazione 0 è sempre la transazione *coinbase*, ovvero una transazione speciale che “genera” nuovi coin. Essa prende tutte le fee implicite delle altre transazioni nel blocco e le distribuisce in output a delle Public Key possedute da dei Miner. Le transazioni possono trovarsi in qualsiasi ordine all'interno del blocco, ad eccezione della transazione *coinbase* che deve essere sempre la prima, ma possono spendere solo gli output di transazioni precedenti ad esse nel blocco o in un altro blocco convalidato della blockchain. [14] Lezione 5

## 1.4 Il Mining

All'interno del network peer-to-peer di Bitcoin troviamo dei calcolatori, i quali avranno attivo il software di Bitcoin, chiamati Nodi. Questi nodi possono essere di vario tipo e attivi per motivi differenti a seconda delle finalità dello stesso. Infatti troviamo dei nodi attivi che gestiscono le transazioni in cambio di fee (Miner) e possiamo anche trovare nodi che gestiscono il sistema del wallet e dei nuovi arrivi (Full Nodes), nodi che collezionano dati sul network ecc. In particolare, analizziamo ora i nodi Miner.

Tutto il sistema di Bitcoin dipende dai cosiddetti Miner. Essi si occupano sia di controllare e convalidare ogni transazione che ricevono in broadcast tramite la rete Bitcoin, che di inserirle all'interno di blocchi appositi, salvandoli in memoria, e decidendo quali blocchi devono essere inclusi nella Blockchain. I Miner portano avanti la rete Bitcoin in cambio di reward, le fee della transazione zero <sup>9</sup>, ottenute ogni volta che un blocco viene convalidato, insieme ad un reward intrinseco per

---

<sup>9</sup>La prima transazione di ogni blocco è la transazione *Coinbase* che distribuisce le fee del blocco ai Miner che hanno svolto la PoW.

ogni blocco creato che viene assegnato dal software stesso di Bitcoin. A grandi linee, il compito di un nodo Miner è quello di rimanere in ascolto per delle transazioni lanciate in broadcast e controllarne la validità, ascoltare per nuovi blocchi della blockchain (mantenendo in memoria l'intera Blockchain) e controllare che essi siano validi. La parte attiva di un nodo Miner Bitcoin è quella di creare un blocco con le transazioni ricevute, che ha convalidato, per continuare la catena di blocchi e portare a compimento quelle transazioni. Dopo aver assegnato il blocco precedente e aver costruito l'albero delle transazioni, deve eseguire la parte più pesante di tutto il processo, ovvero quella di trovare un valore di *Nonce* valido. Questo procedimento consiste nell'assegnare al campo Nonce un valore tale che, facendo passare il blocco insieme al valore di Nonce, l'hash risultante sia numericamente inferiore alla soglia della difficoltà impostata dalla rete (nota che assegna al DAA). Se l'hash ottenuto è numericamente troppo grande rispetto al quel valore, allora bisognerà assegnare un nuovo valore a Nonce e ripetere l'operazione e così via, fino a trovare un valore tale da creare un hash inferiore al limite. Questo procedimento viene anche chiamato Proof of Work che approfondirò nel Capitolo 2.3 pagina 36.

Successivamente, il blocco viene lanciato in broadcast e il nodo rimane in ascolto per sapere se gli altri Miner lo hanno accettato come valido e stanno costruendo i loro blocchi sopra di esso.

Se gli altri nodi cominciano a costruire i loro blocchi sopra il blocco che si è inviato, allora il nodo mittente riceverà dei reward dalla rete, ovvero i pagamenti delle fee della coinbase e un "Block Reward", cioè una ricompensa intrinseca della rete Bitcoin che viene assegnata ogni volta che un blocco viene creato.

Tramite il Block Reward vengono creati dei nuovi coin ogni volta che un blocco viene "minato"; il numero di BTC creati però è sempre inferiore nel tempo ed è de-

stinato ad arrivare a 0 dato che, quando Bitcoin è stato creato, Satoshi ha stabilito un numero massimo di BTC per tutta la rete, ovvero 21.000.000 BTC. Questa soglia verrà raggiunto all'incirca nel 2050, data calcolata con l'attuale difficoltà di creazione dei blocchi.[7]

Approssimativamente ogni 10 minuti viene creato un nuovo blocco di transazioni, quindi in 14 giorni vengono creati circa 2016 blocchi, l'algoritmo di Bitcoin che regola la difficoltà cerca di mantenere stabili questi valori; lo approfondiremo più avanti, nella Sezione 2.4

## 1.5 Decentralizzazione di Bitcoin

Bitcoin è un software nato per essere una moneta alternativa che non fosse controllata da un potere centrale o dalle banche, di conseguenza è stata progettata per essere totalmente decentralizzata e non avere un'autorità centrale. Tutti possono scaricarsi i software necessari per diventare un nodo Bitcoin ed operare nel network peer-to-peer e tutti possono partecipare al mining, anche se con meno efficienza rispetto ad altri; tale problema, che affronteremo nella prossima sessione, si sta facendo sempre più presente. Di conseguenza, non c'è una singola autorità che gestisce il flusso del network. Inoltre, la blockchain non è custodita da nessuno, tutto i nodi possiedono una copia della Blockchain che è totalmente pubblica ed accessibile a tutti coloro che vogliono analizzarla. Bitcoin riesce ad ottenere tutto ciò facendo funzionare il suo network peer-to-peer tramite un consenso distribuito. Non c'è una singola entità a decidere quali transazioni hanno la precedenza e quali no. Tutti i nodi partecipano a questo "gioco" ed è necessario il consenso dell'almeno 51% dei partecipanti per poter modificare la Blockchain <sup>10</sup>. Ciò protegge Bitcoin da attac-

---

<sup>10</sup>Per poter vincere costantemente contro l'intero network, gli attaccanti, dovrebbero riuscire ad ottenere il 51% dell'Hash Power della rete. In questa maniera potranno superare, costantemen-

chi di nodi maligni che vogliono modificare la storia della blockchain, o i successivi blocchi, a loro favore; per riuscirci, dovrebbero ottenere il 51% del potere di calcolo totale di tutta la rete Bitcoin.[8]

### 1.5.1 Tendenza verso la Centralizzazione

Anche se Bitcoin è nato come sistema completamente decentralizzato, negli ultimi tempi si è vista una continua tendenza alla centralizzazione, o alla supremazia, di alcuni suoi aspetti e parti all'interno del processo. Moltissimi dei calcolatori, che sono occupati a lavorare nel network Bitcoin, provengono da aziende private, tra cui l'azienda Bitmain<sup>11</sup> che prenderemo in considerazione per l'esempio. Nonostante Bitmain provveda solo a fornire tecnologia ASIC<sup>12</sup>, quindi non ne gestisce personalmente l'uso, potrebbe comunque prendere decisioni sui propri dispositivi, in futuro, per il tornaconto dell'azienda stessa, influenzando di molto la rete Bitcoin insieme al suo Hash Rate. Sempre inerente al discorso dell'Hash Rate, nei tempi si sono formate delle Pool di Miner<sup>13</sup> estremamente grandi e che hanno attirato a sè tantissimi Miner, con esse, la maggior parte dell'Hash Rate dell'intera rete Bitcoin si è “centralizzato” sotto una stessa Pool.

---

te, il resto della rete nella creazione del Nonce dei blocchi, mandando in broadcast i loro blocchi modificati

<sup>11</sup>Di seguito, il link alla pagina web dell'azienda: [www.bitmain.com](http://www.bitmain.com)

<sup>12</sup>Application Specific Integrated Circuit, sono dei calcolatori specializzati nell'esecuzione ripetitiva di un determinato compito, nel nostro caso il calcolo del Nonce. Sono generalmente estremamente potenti e veloci nello svolgere il proprio compito, ma non possono essere utilizzati per altro.

<sup>13</sup>Il pool mining è l'atto di unire le risorse da parte dei minatori, che condividono la loro potenza di elaborazione su una rete, per dividere equamente la ricompensa, in base alla quantità di lavoro che hanno contribuito alla probabilità di trovare un blocco

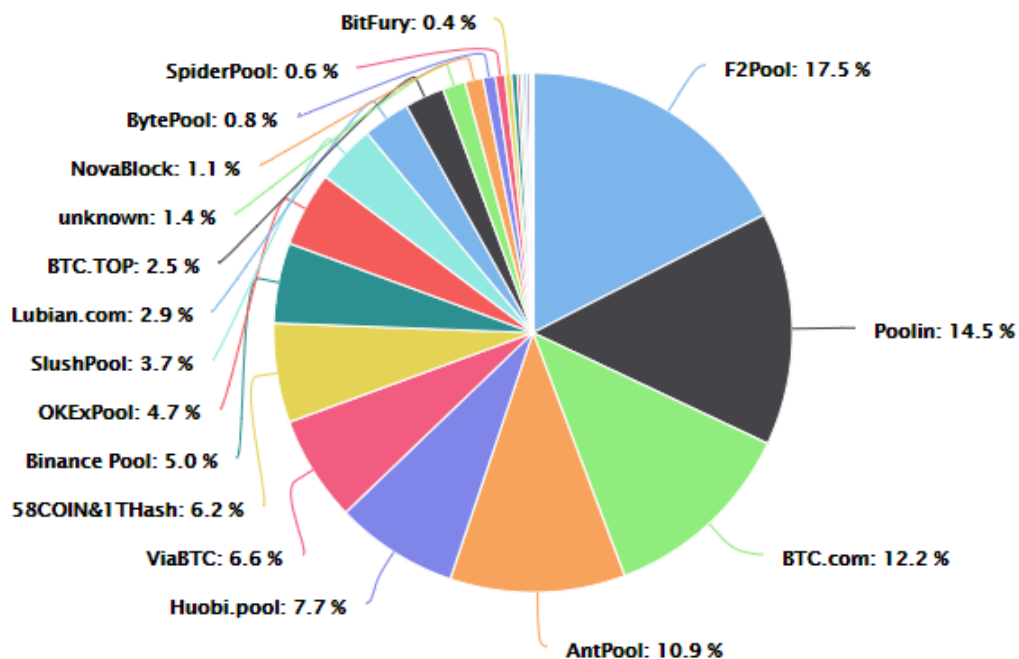


Figura 1.7: Grafico dell'Hash Rate del giorno 14/01/2021 con prospettiva annuale

Ciò non è di per sé un problema, ma fa spazio alla possibilità che, queste pool di Miner possano accordarsi ed unire il proprio Hash Rate per effettuare un attacco del 51% manipolando così, a proprio piacere, le transazioni e la storia di Bitcoin[4]. Questo si avvicina molto alla realtà dato che, nella metà del 2014, la Pool di *Miner GHash* riuscì a raggiungere quota oltre il 50% dell'Hash Rate totale della rete di quei tempi, cosa che fece tremare la fiducia del sistema stesso. Fu la Pool stessa a mettere un freno agli avvenimenti, limitando il proprio Hash Rate ad un massimo di 39.99%; ma questo evento è comunque un tassello che ci permette di analizzare come, con il tempo, anche Bitcoin potrebbe correre dei rischi per colpa della centralizzazione dell'Hash Rate. Inoltre, c'è anche il problema della centralizzazione dell'Hash Rate a livello politico. Infatti attualmente l'80% circa dell'Hash Rate dell'intera rete proviene dalla Cina e solo il 20% dal resto del mondo. Ad oggi Bitcoin



utilizza una Proof of Work per gestire la propria rete.

Alcune Criptovalute, come Ethereum <sup>14</sup>, stanno piano a piano abbracciando questi tipi di Proof per abbandonare la PoW, si tratta di un discorso estremamente lungo e complesso.[9]

---

<sup>14</sup>Ethereum è una piattaforma Open Source Global per applicazioni decentralizzate con una propria criptovaluta, l'ETH. Nata per funzionare tramite una PoW, è attualmente in fase di transizione da PoW a PoS. Secondo la tabella di marcia dovrebbe abbandonare completamente la PoW entro fine 2021. Di seguito il link alla pagina principale: [ethereum.org](https://ethereum.org)

## Capitolo 2

# Blockchain: il public ledger di Bitcoin

Una delle componenti principali di Bitcoin è il suo Public Ledger, conosciuto anche con il nome di Blockchain, dove tutte le transazioni mai avvenute all'interno del network di Bitcoin vengono registrate e salvate dentro questo "database". Nelle istituzioni finanziarie moderne, questo ledger è gestito dall'istituzione stessa, di conseguenza l'user che ne usufruirà dovrà avere fiducia nell'istituzione e nella sua regolamentazione del processo dato che, molto spesso, le procedure vengono effettuate con l'utente all'oscuro dei vari micromovimenti. Inoltre, la transazione deve essere inoltrata ad ogni nodo della rete che la riceverà e ne terrà una copia in memoria, dopo aver controllato da se che la transazione sia effettivamente valida. L'innovazione di Bitcoin sta nel semplice fatto che, ogni utente, può creare la propria transazione e firmarla totalmente in autonomia e solo con l'utilizzo della sua coppia di chiavi (Secret & Public), essendo così totalmente in controllo dei propri fondi. All'interno di questo sistema non c'è niente di privato dato che tutti possono vedere le transazioni che avvengono all'interno della rete, ma nessuno può ricollegarle ad una identità reale tramite solo queste informazione. [7]

## 2.1 La Catena di Bitcoin

Bitcoin si contrappone alle istituzioni attuali rendendo totalmente pubblico la blockchain, insieme ad ogni singola transazione eseguita all'interno della rete peer-to-peer.

La blockchain di Bitcoin non è altro che un ledger pubblico, condiviso e aggiornato da tutto il network e dai suoi membri. A questo Ledger viene aggiunta, globalmente, una pagina circa ogni 10 minuti e, essendo pubblico, tutti i possessori del software di Bitcoin possono averci accesso costantemente dato che ne custodiranno una copia, da mantenere aggiornata, all'interno del proprio dispositivo.

Nel contesto specifico di Bitcoin, questo Ledger pubblico è chiamato, per l'appunto, "Blockchain" e ogni pagina di questo Ledger rappresenta un "blocco" di dati. All'interno di ogni blocco troviamo una serie di transazioni e l'insieme di tutti i blocchi crea, per l'appunto, una catena di blocchi.

Questa "catena" aumenta costantemente la sua dimensione e, in essa, possiamo trovare tutte le transazioni che sono avvenute nella rete Bitcoin, dal gennaio 2009 fino al momento della sua lettura. Tramite questa catena, i Miner riescono a tenere costantemente traccia dei fondi disponibili in ogni indirizzo Bitcoin. Grazie ad un protocollo di consenso distribuito, i vari nodi mantengono le proprie repliche della blockchain aggiornate e consistenti.[4]

Questi indirizzi della rete Bitcoin sono pubblici, di conseguenza chiunque può calcolare quanti fondi ci sono all'interno di un determinato indirizzo, ma essi sono completamente scollegati dall'identità dei loro possessori dato che, Bitcoin, non utilizza nessuna informazione personale dell'User, ne richiede qualche tipo di prova d'identità; questi indirizzi sono soltanto delle lunghe stringhe di caratteri e numeri.[7]

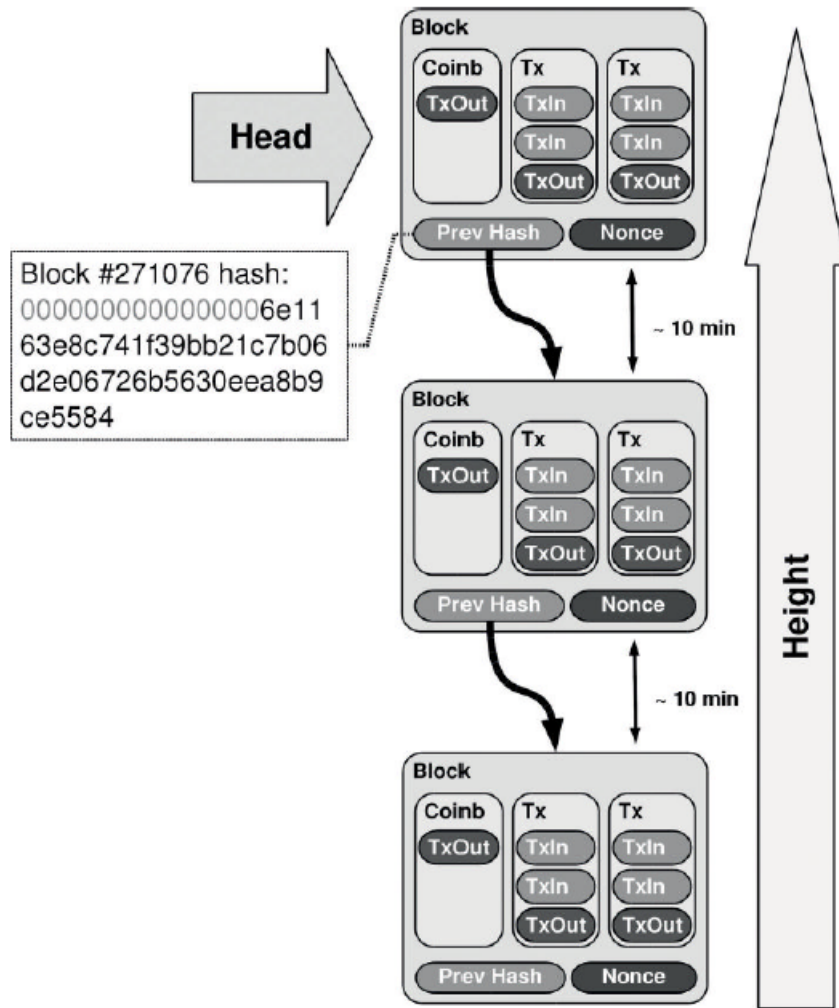


Figura 2.1: Esempio logico di blocchi collegati tra loro all'interno della blockchain

Tutta la Blockchain è formata da blocchi con al loro interno un campo “Prev Hash” che fa riferimento all’Hash del blocco precedente ad esso. Ogni blocco ha ben definito il blocco dietro di lui, ciò fa sì che la Blockchain sia resistente ai cambi dato che, manomettere anche solo un singolo bit all’interno di quel blocco, farebbe sì che l’intero hash del blocco stesso cambi<sup>1</sup>. Se l’hash di un blocco cambia, allora il blocco successivo farà riferimento ad un blocco inesistente, rendendo ovvia la modifica e il tentativo illecito. Questo meccanismo fa sì che la blockchain sia estremamente

<sup>1</sup>Noto come effetto Avalanche, ne abbiamo discusso in precedenza nel 2

resistente a questa tipologia di attacchi dato che si noterebbero subito un cambiamento; inoltre tutti i nodi della rete peer-to-peer possiedono una copia intatta della blockchain nella loro memoria.[7]

## 2.2 Accenno alle Fork

Durante la vita di Bitcoin, il suo software è stato aggiornato costantemente dalla sua community, cambiando il suo sistema originale. Essendo il progetto Bitcoin Open Source, alcuni dei vari aggiornamenti al codice esistente non sono stati ben accettati e, a volte, non sono stati condivisi dai vari utenti del sistema che hanno, volontariamente, scelto di non aggiornare il proprio software e rimanere con la versione precedente.

Definiamo una fork come l'evento di una diramazione della blockchain in due o più catene. Esse avvengono nel caso in cui un aggiornamento crei una biforcazione tra software aggiornato e software non aggiornato, ma possono crearsi anche per puro caso nella normale vita della blockchain stessa, anche senza un aggiornamento del sistema.

Ciò accade quando due blocchi differenti puntano allo stesso blocco predecessore, cosa che può accadere molto spesso quando due Miner differenti trovano, in un lasso di tempo molto ristretto, dei blocchi validi che lanciano in broadcast. Nel caso di fork nate da se, questo problema non richiede una soluzione attiva della community dato che, prima o poi, una delle due chain risulterà più lunga rispetto all'altra e la politica di Bitcoin è quella di considerare valida la chain che ha più "lavoro" su di se. Ovvero si darà più credito alla chain più lunga nel tempo dato che sarà anche la chain che ha più PoW su di essa<sup>2</sup>.

---

<sup>2</sup>Mettendo come premessa che tutta la rete è disposta ad accettare l'aggiornamento, avendo

Nel caso di fork dovute ad aggiornamenti, a seconda dell'importanza, delle novità e dei cambiamenti stessi che porta nel codice, si parla di Hard Fork o di Soft Fork, Fork che creano un effetto differente sulla Blockchain.

In ogni caso, una fork, se risolvibile, dovrebbe essere gestita in breve tempo dato che si “spreca” calcolo nel portare avanti due chain differenti in parallelo.

### 2.2.1 Soft Fork

Gli aggiornamenti che caratterizzano una Soft Fork sono degli aggiornamenti “retro-compatibili”, ovvero che restringono un certo set di regole della versione precedente. Durante questo aggiornamento si modificano le regole in modo tale che tutti i nodi  $V_1$  che non hanno eseguito l'aggiornamento vedano i blocchi creati dai nodi  $V_2$ , che hanno eseguito l'aggiornamento, come validi. Per esempio, se un aggiornamento consistesse nel modificare la size dei blocchi creati dai Miner da 1MB a 0.5MB, tutti i nodi  $V_1$  che non hanno effettuato l'aggiornamento vedranno i blocchi di max 0.5MB creati dai nodi aggiornati  $V_2$  come validi, ma non è vero il contrario. Le proprietà intrinseche delle Soft Fork sono ottime per il periodo di transizione dato che, chi non ha effettuato l'upgrade per motivi temporali, non si ritroverà a “sprecare” il proprio hash power e la rete Bitcoin ne risentirà, in generale, meno del peso dell'aggiornamento.

---

un Hash Power totale, sulla biforcazione della catena che ha effettuato l'aggiornamento, sempre maggiore nel tempo, essa diventerà più lunga dell'altra biforcazione dato che, con più Hash Power, è più consistente, per la rete, trovare un Nonce valido per i propri blocchi.

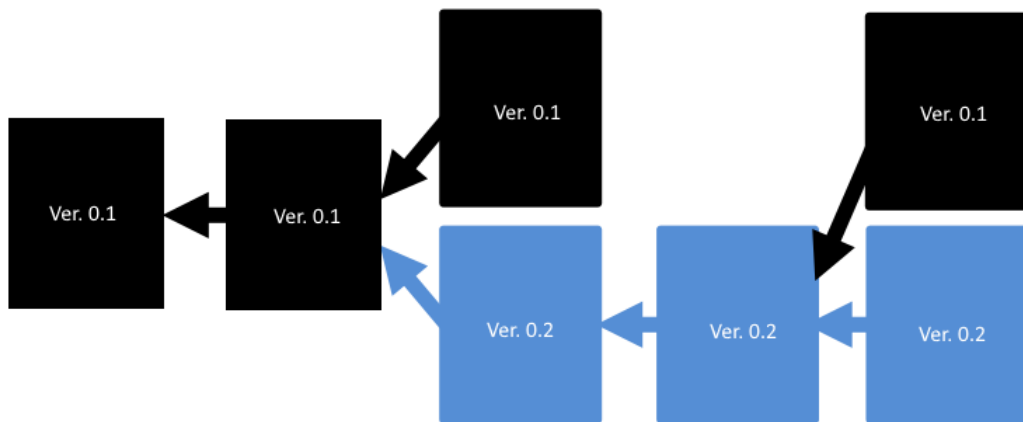


Figura 2.2: Esempio Logico di una Soft Fork

Nel tempo, tutti i nodi  $V_1$  senza aggiornamento creeranno dei blocchi che diventeranno orfani, ovvero che, con il passare del tempo, verranno abbandonati dalla rete. Con il tempo l'Hash Rate passerà alla catena sviluppata dai nodi  $V_2$  dato che, piano a piano, la rete si aggiornerà se accetterà il cambiamento.[14]

### 2.2.2 Hard Fork

Al contrario, durante una Hard Fork, gli aggiornamenti che vengono attuati non sono retrocompatibili. Saranno nuove regole meno restrittive che faranno sì che tutti i nodi  $V_1$  che non effettueranno l'aggiornamento vedano i blocchi creati dai nodi  $V_2$ , che hanno effettuato l'aggiornamento, come invalidi. Le Hard Fork creano la possibilità di una diramazione persistente della Blockchain nel caso in cui l'aggiornamento non venga accettato da molti utenti del network.

Questo perché chi non accetterà i cambiamenti non vedrà le nuove regole come valide e continuerà a costruire una catena di blocchi valida solo per i nodi che non hanno aggiornato. Viceversa, i nodi che hanno aggiornato creeranno una catena di nodi a se che non potrà mai essere riconosciuta dai nodi che non hanno effettuato

l'aggiornamento. Quindi se l'aggiornamento viene rifiutato e i nodi che non hanno aggiornato non lo faranno mai, la Blockchain si vedrà divisa in due con nessuna possibilità di ricongiungimento. [14]

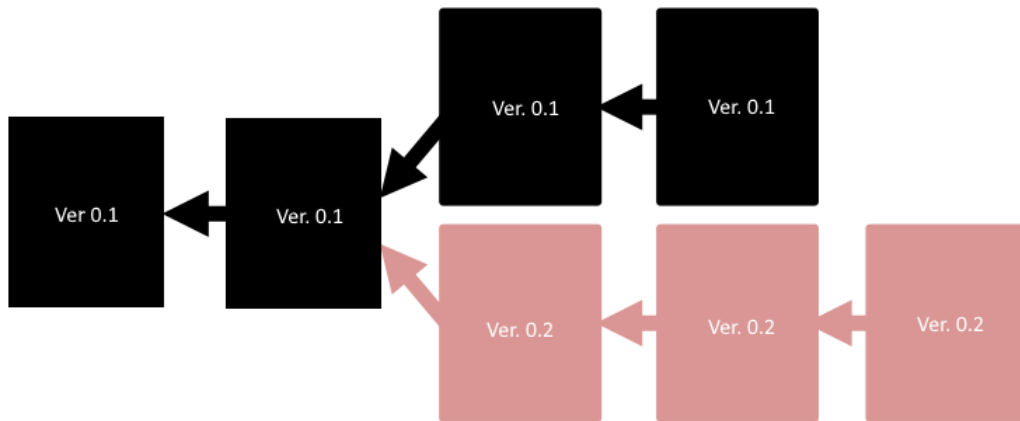


Figura 2.3: Esempio Logico di una Hard Fork

### 2.2.3 Esempi di Fork

Per un esempio di Hard Fork di Bitcoin possiamo prendere in considerazione Bitcoin Cash, criptovaluta creata dopo una fork di Bitcoin nel 2017 e successivamente, nel 2018, Bitcoin Cash è stato sottoposto ad un'altra fork, risultando nella divisione in Bitcoin Cash e Bitcoin SV. La fork del 2017 nacque perché programmatori attivi di Bitcoin sentirono la necessità di renderlo più veloce e scalabile<sup>3</sup>, creando nuove regole per la blockchain e aggiustandone alcune, come la grandezza dei blocchi. I sostenitori di Bitcoin che accettavano la nuova direzione e visionavano un uso commerciale quotidiano della moneta sostenevano il bisogno di aumentare la grandezza, in MB, di un singolo blocco della Blockchain, permettendo così di aumentare le transazioni al secondo disponibili per la convalida all'interno dell'intero network

---

<sup>3</sup>Attualmente Bitcoin può gestire fino ad un massimo di 7 transazioni al secondo, cosa che non lo rende utilizzabile come moneta digitale per le masse. Questo limite è dovuto anche dalla regola per il limite di grandezza dei blocchi creati dai Miner.



mentre, i sostenitori di Bitcoin originale come ideato da Satoshi, non volevano portare la valuta ad essere utilizzata in questa maniera, sostenendo i vecchi ideali con cui la moneta era nata. Questa differenza di opinioni portò la catena di blocchi a biforcarsi nel 2017, creando così Bitcoin Cash.

Questo è solo uno dei tanti esempi di Fork che il sistema di Bitcoin, come anche altri sistemi di criptovalute, ha subito nell'arco della sua vita.

## 2.3 La Proof of Work

La Proof of Work (PoW) è una forma di crittografia zero-knowledge proof<sup>4</sup> dove una parte chiamata “Fornitore” (nel nostro caso il singolo Nodo) prova ad una seconda parte, chiamata “Verificatore”(la Rete peer-to-peer) che, per un determinato motivo (trovare il valore di Nonce) è stata impiegato un certo effort computazionale.

Questo concetto è stato inventato da Cynthia Dwork e da Moni Naor nel 1993[11], all'epoca era un metodo per difendersi e scoraggiare dei malintenzionati, nel fare un attacco ad una rete, tramite del lavoro computazionale. Questo metodo è stato abbracciato da Satoshi con il progetto di Bitcoin, rendendolo uno dei pilastri della rete.

All'interno della rete Bitcoin, un Miner deve risolvere un problema associato ad un blocco se vuole inoltrarlo in broadcast. Questo lavoro è per l'appunto la PoW ed implica che il Miner in questione ha dovuto lavorare per trasmettere il blocco. [4]

---

<sup>4</sup>La Zero-Knowledge Proof è una tecnica della crittografia dove una parte, detta "Fornitore", può provare ad un'altra parte, detta "Verificatore", che conosce un determinato valore  $x$  senza dover diffondere altre informazioni a riguardo se non il fatto di sapere il valore di  $x$ . In altre parole la Zero-Knowledge Proof deve essere provabile, da chi possiede le informazioni, semplicemente rivelando il fatto che la si conosce.

### 2.3.1 Il Mining e la Proof of Work

Come abbiamo accennato in precedenza, la PoW di Bitcoin consiste nell'utilizzare il potere computazionale di un nodo per risolvere un "puzzle".

Per poter inviare nella rete tramite broadcast il proprio blocco appena creato, un nodo deve prima di tutto creare il blocco tramite le transazioni valide che ha ascoltato, prendere l'Hash dell'ultimo blocco all'interno della Blockchain e assegnare un valore al campo Nonce. Successivamente il nodo dovrà concatenare queste informazioni e farle passare per la funzione di Hashing.

L'output che si otterrà effettuando l'hash di queste informazioni, per dichiarare valido il blocco, dovrà rientrare all'interno di un determinato range estremamente stretto rispetto a tutti i possibili output, valore range scelto tramite l'algoritmo DAA che discuteremo brevemente nella Sezione 2.4.

$$H(\text{nonce}|\text{prevHash}|\text{merkleRoot}) < \text{targetRange}$$

Se il valore assegnato al Nonce non creerà un Output valido per il blocco, allora si cambierà quel valore e si controllerà il nuovo Output. Questo procedimento proseguirà in loop finché il nodo non riuscirà ad ottenere un valido risultato, oppure un altro nodo all'interno della rete lo farà prima di lui.

Il lavoro richiesto per trovare questo Nonce valido è estremamente dispendioso a livello di calcoli e non è qualcosa che possa essere fatto velocemente.

È proprio questa l'essenza della PoW, il nodo deve utilizzare molta potenza di calcolo per trovare un valore di Nonce valido e lo farà competendo con l'intero network peer-to-peer che cercherà di fare la stessa cosa con i propri blocchi.

Prima o poi, all'interno della rete, qualcuno troverà un Nonce che risolverà il puzzle e il blocco creato da quel nodo andrà a fare parte della Blockchain, in questa manie-

ra la rete è completamente decentralizzata ed ogni nodo è in competizione con tutta la rete. Nessuno può decidere chi proporrà il successivo nodo dato che solo il lavoro di ricerca del Nonce potrà definirlo[7].

Abbiamo appena discusso della difficoltà di trovare un valore di Nonce valido, ma c'è anche da dire che questa difficoltà non è presente nell'altra direzione. Ovvero la verifica del risultato ottenuto da un nodo è effettuata, prima di inserire il blocco nella propria copia della blockchain, da tutti i nodi della rete peer-to-peer ed è un'operazione banale da calcolare, al contrario della sua controparte. Dati i valori delle variabili del blocco, controllare il risultato della funzione di Hashing è estremamente veloce, detto ciò possiamo dire che controllare una soluzione non sia pesante a livello temporale e di calcolo per la rete, quindi la PoW si limita semplicemente alla ricerca di questi valori, ovvero alla risoluzione del puzzle e non al controllo della soluzione.

La Proof of Work, insieme al sistema della Blockchain, rende estremamente difficile attaccare Bitcoin o la blockchain stessa dato che, il malintenzionato dovrebbe modificare tutti i blocchi seguenti ad un blocco per poter rendere il blocco modificato accettabile dato che, una singola modifica del sistema, creerebbe un effetto a cascata su tutti i blocchi successivi. (effetto avalanche) Inoltre, nuovi blocchi sono “minati” ogni 10 minuti, di conseguenza il numero di blocchi da modificare aumenterebbero con il tempo, rendendo ancora più difficile l'impresa.[14]

## 2.4 Difficulty Adjustment Algorithm

Con il passare del tempo dalla creazione del software Bitcoin, le tecnologie si sono evolute e, con loro, i calcolatori utilizzati dagli User per effettuare la PoW. Queste nuove tecnologie avrebbero dovuto aumentare di molto la velocità di creazione di un

blocco, passando da una media di 10 minuti a pochi secondi, ma ciò non è avvenuto e questo è dovuto al fatto che Bitcoin ha implementato al suo interno un algoritmo che, prendendo alcuni parametri della rete, aggiusta in modo regolare e autonomo la difficoltà che incontrano i Miner nel creare un Nonce valido per i propri blocchi, mantenendo così costante il tempo medio impiegato per la convalida di un singolo blocco tramite il Nonce.

Quindi Bitcoin regola il proprio network in modo tale che i blocchi nella rete vengano creati in maniera periodica. L'algoritmo *Difficulty Adjustment Algorithm* (DAA) è colui che si occupa di questo compito; la difficoltà di mining viene aggiornata ogni 2016 blocchi in accordo alla formula:

$$\text{nextDifficulty} = (\text{previousDifficulty} * 2016 * 10) / (\text{timeToMineLast2016Blocks})$$

Formula che permette al sistema di aggiornare la difficoltà in base all'efficienza media dei Miner che, in quei 2016 blocchi, hanno minato dalla volta precedente. Contando che un blocco viene creato ogni 10 minuti circa e che questo controllo avviene ogni 2016 blocchi, possiamo dire che ogni 14 giorni circa, questo aggiornamento della difficoltà prende atto, cercando di mantenere l'equilibrio del sistema, bilanciando la difficoltà in modo tale che i blocchi continuino ad uscire ogni 10 minuti circa.[8]

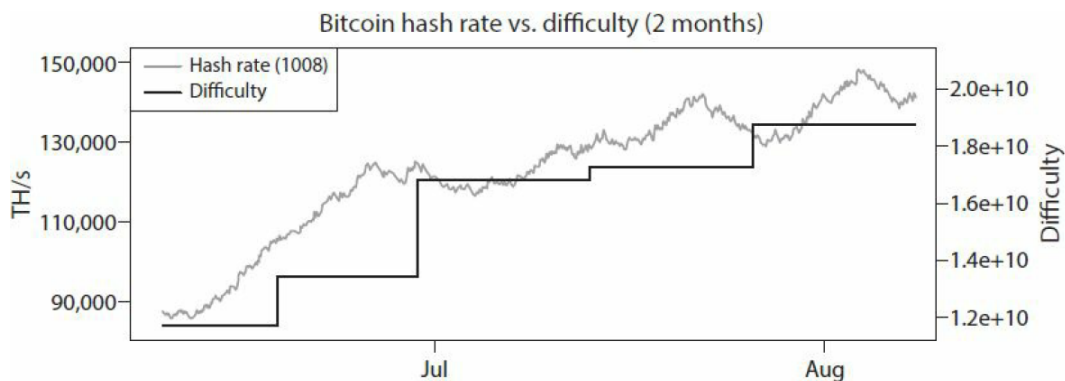


Figura 2.4: Hash Rate del 2016 –“bitcoinwisdom.com”

Nel tempo la difficoltà tende ad aumentare, complice anche lo sviluppo di calcolatori creati appositamente per il Mining e che raggiungono degli Hash Rate sempre maggiori, per cercare di rimanere competitivi nel mercato. Questa crescita della difficoltà dipende anche dal numero totale di Miner che ci sono all'interno della rete, motivo per cui nuovi Miner non sono visti di buon occhio dai Miner più "anziani"[8].

## Capitolo 3

# Introduzione a Blockcerts

Come abbiamo visto, la blockchain è uno strumento duraturo, distribuito e resistente a gli attacchi di agenti maligni. Ciò la rende estremamente efficace e sicura come mezzo per poter salvare, online e in un modo completamente decentralizzato, tutte le transazioni avvenute all'interno del network Bitcoin.

Nonostante la blockchain sia stata ideata per essere un puro ledger pubblico per la moneta di Bitcoin, sfruttando le UTXO stesse di Bitcoin, è possibile anche utilizzarla per inserire dei dati, non inerenti alla semplice transazione, e memorizzarli nella Blockchain stessa. Queste informazioni saranno fatte passare all'interno di più funzioni crittografiche di Hash, rendendole, poi, pubbliche<sup>1</sup>. Questo procedimento può essere svolto con un qualsiasi tipo di dato.

Mostreremo di seguito come, questa idea, viene applicata dal progetto Blockcerts come uno standard per creare, emettere, visionare e verificare dei certificati, di un qualsiasi settore e in maniera totalmente “trustless”, tramite la tecnologia della blockchain.

---

<sup>1</sup>Dato che la blockchain è totalmente pubblica, ed accessibile a chiunque faccia parte del network, chiunque potrà visualizzare le informazioni inerenti alla transazione

## 3.1 Il progetto Blockcerts

Blockcerts è in progetto open source<sup>2</sup> che sfrutta tutta la tecnologia di bitcoin e della blockchain appena discussa per creare un metodo per firmare dei certificati all'interno di un ambiente blockchain. È stato ideato e realizzato dal MIT Media Lab, anche se non sono coinvolti attivamente con lo sviluppo del progetto attualmente, e da Hyland Credentials che, invece, sta portando avanti il progetto. Esso consiste in una libreria open source, vari strumenti per l'applicazione dei certificati e una App per telefoni che permette di verificare, in totale autonomia, i certificati registrati. L'idea dietro blockcerts è quella di registrare l'hash di questi certificati all'interno di una transazione che verrà inviata in broadcast nel network. In questo modo, i miner la inseriranno all'interno di un loro blocco di cui poi cercheranno il nonce. Nel momento in cui uno di questi blocchi diventerà valido, la transazione diventerà parte integrante della blockchain e, di conseguenza, godrà delle stesse proprietà delle transazioni bitcoin<sup>3</sup>.

Tramite questo progetto, un utente ha la possibilità di condividere, a chiunque, tutti i propri certificati ancorati alla blockchain, senza nessun intermediario, senza spese di denaro o, soprattutto, di tempo.

Tutti i certificati ancorati alla Blockchain potranno essere condivisi online senza nessuna preoccupazione, essi saranno sicuri e protetti da ogni tipo di manomissione dato che, come le transazioni e i blocchi, ogni tipologia di modifica ad essi comporterà un effetto Avalanche su tutti i blocchi successivi al blocco contenente il certificato. Di conseguenza, chiunque cerchi di verificare un determinato certificato potrà

---

<sup>2</sup>Open Source indica che è in progetto totalmente pubblico e non protetto da copyright dove, ogni linea di codice, aggiornamento ecc è tranquillamente consultabile e modificabile da chiunque.

<sup>3</sup>Ovvero quelle di essere pubbliche, crittografate, potenzialmente impossibili da manomettere e di poter essere lette tramite delle chiavi.

essere sicuro della sua integrità e non dovrà preoccuparsi di un possibile falso.[3]. Inoltre, solo chi in possesso delle credenziali del certificato<sup>4</sup> potrà visualizzarlo tramite un processo di verifica. Tutto ciò senza interpellare l'istituzione che ha emesso il certificato, dato che dovrà solo creare le credenziali di verifica una volta. [10]

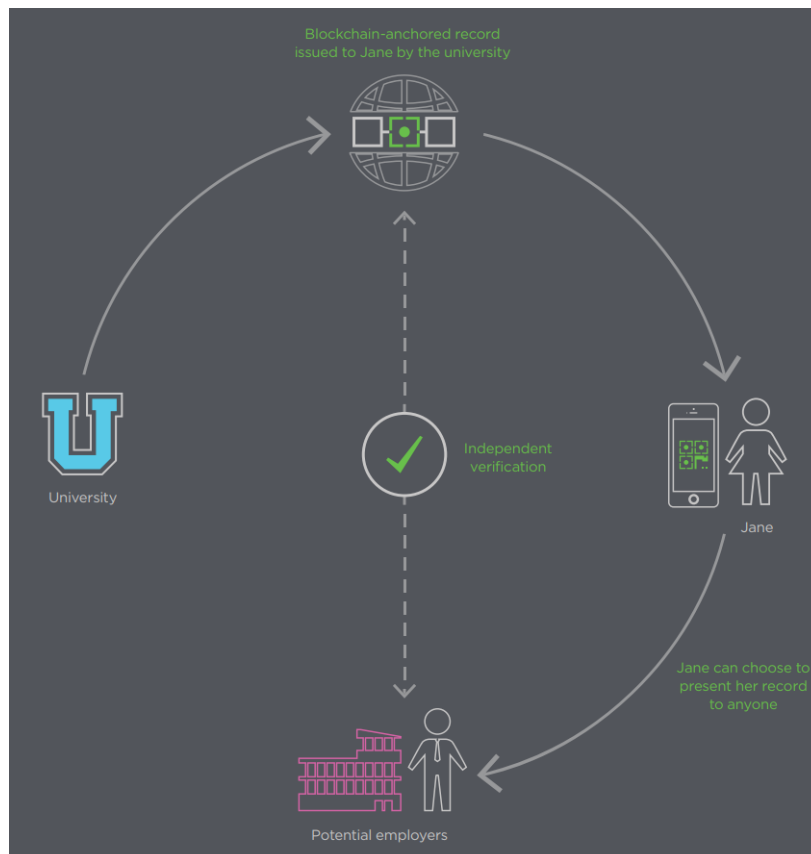


Figura 3.1: Esempio logico di un caso d'uso [10]

Nello scenario della Figura 3.1 la studentessa Jane si è appena laureata. Riceve dall'università una copia digitale ufficiale del suo diploma che possederà da quel momento in poi. Ora potrà presentare il proprio certificato di diploma a qualsiasi istituzione, o ad un datore di lavoro, che verificherà autonomamente l'emittente del

<sup>4</sup>Credenziali che verranno emesse dall'istituto. Verranno create insieme all'emissione del certificato nella blockchain.



diploma e lo status di esso<sup>5</sup>

Questa operazione limiterà il lavoro dell'università, riducendo a pochissimo il tempo necessario per verificare il diploma stesso; si eliminerà anche la carta da questo processo, eliminando così la possibilità di una possibile copia falsa del diploma, riducendo anche i costi della stampa.[10]

## 3.2 Esempio di caso d'uso

Dopo aver emesso un qualsiasi tipo di certificato, l'istituto potrà chiedere al beneficiario se vuole che venga inserito all'interno della Blockchain. In caso di una risposta positiva, il beneficiario dovrà fornire un address bitcoin, di cui è in possesso, all'istituto. Successivamente l'istituto userà la propria firma digitale, il certificato in questione e la Public Key fornita dal beneficiario come Input per una funzione di hash crittografica. Il risultato verrà inserito all'interno della Blockchain tramite una transazione.

Alla fine di questo procedimento, l'istituto passerà le credenziali al beneficiario. Queste credenziali potranno essere condivise con chiunque e in qualsiasi momento dal beneficiario stesso; il datore di lavoro che si troverà davanti queste credenziali potrà, in totale autonomia, verificare il certificato all'interno della blockchain.

---

<sup>5</sup>Se attivo, bloccato, revocato ecc.

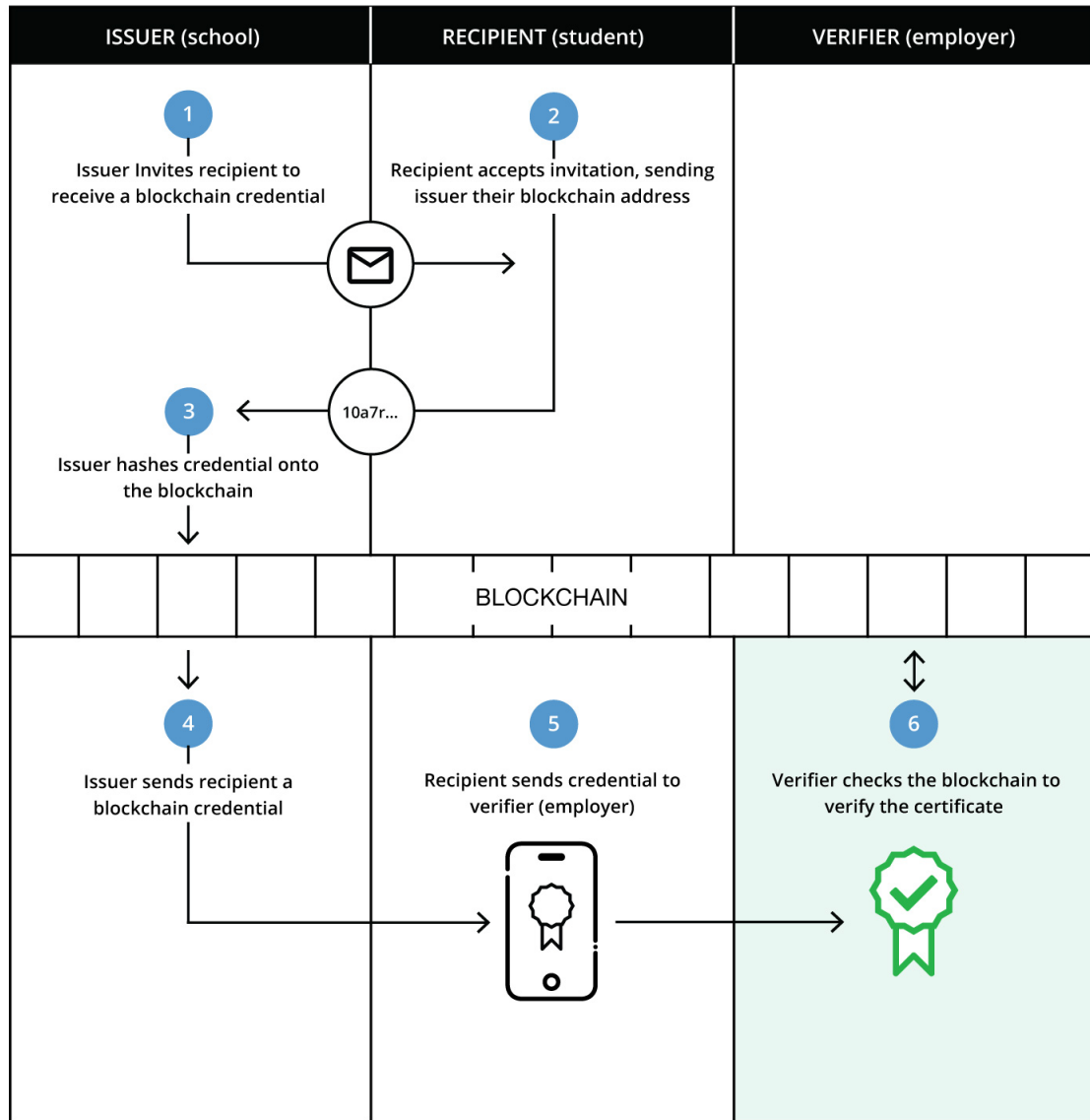


Figura 3.2: Schema logico dei passaggi[3]

Come abbiamo appena visto, dal momento in cui l'istituto emittente consegna le credenziali all'user, non dovrà più far parte del processo di verifica. Ciò permette un processo più veloce e rende l'user in totale possesso, e in controllo, del suo certificato.

### 3.3 Il processo di Blockcerts

Il processo che permette l'inserimento e la verifica dei certificati all'interno del network bitcoin si basa sulle operazioni delle UTXO, nello specifico stiamo parlando dell'operazione `OP_RETURN`. Tramite questo script è possibile inserire 80 bytes arbitrari come parte della transazione, questo permette a chiunque di inserire, all'interno della propria transazione, dei piccoli dati che saranno salvati all'interno della blockchain insieme alla transazione. Il progetto di Blockcerts sfrutta lo script `OP_RETURN` per poter inserire, all'interno della blockchain di bitcoin, i propri certificati.

Il processo di firma dei certificati parte con l'assunzione che, anche se è possibile creare una transazione per ogni singolo certificato che si vuole firmare, la soluzione migliore è quella di firmare più certificati con una sola transazione. Blockcerts ottiene ciò costruendo un Merkle Tree<sup>6</sup> formato dall'hash dei vari certificati che si vogliono firmare tramite quella transazione. Dopo la sua costruzione, la Merkle Root di questo albero, verrà inserita all'interno del campo `OP_RETURN` e la transazione verrà inviata, in broadcast, al network di bitcoin. Di seguito, mostriamo tramite la figura la struttura tipica di una transazione per la firma dei certificati.

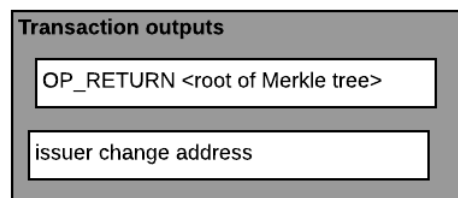


Figura 3.3: Transazione di Blockcerts per la firma dei certificati

Questa transazione avrà in input il minimo numero di bitcoin, presi dal portafoglio

---

<sup>6</sup>L'argomento Merkle Tree è stato approfondito nel capitolo 1.6.

dell'emittente, per poterla rendere valida alla rete. In output troveremo il campo OP\_RETURN con il Merkle Root e, opzionale, il cambio dell'address dell'emittente.

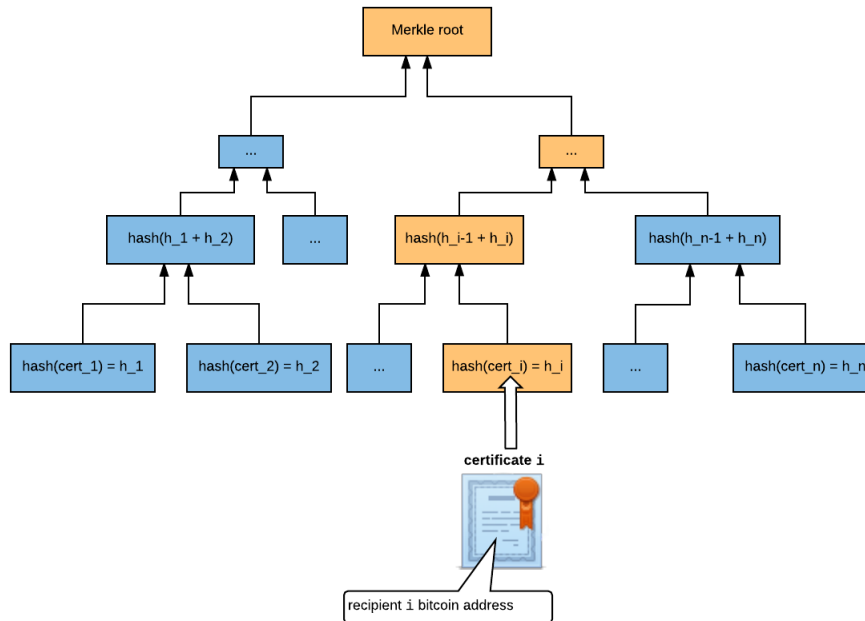


Figura 3.4: Esempio di costruzione di un merkle tree con l'hash dei certificati

La root di questo Merkle Tree sarà un hash di 256 bit formato dalle varie concatenazioni degli hash dei certificati. Ora, se la struttura del Merkle Tree fosse come quella di Bitcoin, tutto ciò non ci aiuterebbe dato che non possiamo sapere se il nostro certificato sia all'interno del Merkle Tree di quella transazione senza avere l'hash di tutti gli altri certificati, o l'hash dei vari nodi padre. Per questo motivo il certificato, che viene creato secondo lo standard Blockcerts, conterrà una firma nel formato "2017 Merkle Proof Signature Suit"<sup>7</sup>.

<sup>7</sup>Questo formato non verrà approfondito all'interno di questa tesi. Se si volesse approfondire l'argomento, consiglio la lettura dell'articolo Merkle Proof Signature Suite 2017.

Il certificato dato al beneficiario conterrà, seguendo questo standard, i seguenti campi.

- L'ID della transazione Bitcoin che contiene la Merkle Root
- Il Merkle Root che bisogna aspettarsi dalla transazione
- L'hash che ci si aspetta per il certificato del beneficiario
- Il percorso, nella figura 3.4 in arancione, dal certificato del beneficiario alla Merkle Root

Il processo di verifica<sup>8</sup> del certificato controlla che :

- L'hash del certificato del beneficiario, all'interno del Merkle Tree, corrisponda al valore sul certificato emesso
- Il path che ci si aspetta sia valido
- Il Merkle Root all'interno della blockchain corrisponda con il valore all'interno del certificato emesso

Questi step permettono di controllare se un certificato sia presente nella Blockchain, se è stato manomesso nel tempo e che non ci siano stati cambiamenti all'interno della blockchain. In altre parole, possiamo confermare che il certificato non sia stato contraffatto o modificato dal momento della sua firma.

Per quanto riguarda l'autenticità del certificato, dobbiamo far affidamento al Bitcoin address che ha creato la transazione. L'issuer che utilizza questo processo dovrebbe inserire il proprio address Bitcoin all'interno, per esempio, della sua pagina

---

<sup>8</sup>Il processo di verifica non verrà approfondito all'interno di questa tesi, per chiunque volesse approfondire l'argomento, consiglio la lettura dell'articolo "Verification Process" all'interno della repository GitHub del team di Blockcerts.

web personale. Ovvero, un luogo pubblico che possa essere modificato solo da lui. In questa maniera, chi vuole verificare l'autenticità del certificato, potrà controllare l'address nelle credenziali e vedere se combacia con l'address pubblico nella pagina web dell'issuer.

### 3.4 Creazione, emissione e visione di certificati locali

In questa sezione mostreremo, passo passo, come sia possibile preparare un ambiente Linux, utilizzando una Macchina Virtuale<sup>9</sup>, e Docker<sup>10</sup> per poter ospitare il processo, appena descritto, di creazione, emissione e visione di un certificato personale. Il tutto all'interno di una Regtest Bitcoin<sup>11</sup>. Questo procedimento ci permetterà di prendere familiarità con la libreria “cert-issuer”, chiave di questo progetto, firmando un modello di certificato senza dover utilizzare Bitcoin Core<sup>12</sup>, ma passando, invece, per Docker.

Illustriamo ora le varie librerie e i processi che ci permettono di creare l'ambiente sopra descritto. Il tutto verrà eseguito tramite l'ausilio di GitHub per le varie librerie, di Docker per la creazione di un ambiente, e di un container, per firmare i nostri certificati, all'interno di un ambiente Linux<sup>13</sup>, e di alcuni modelli di certificati forniti da Blockcerts.

---

<sup>9</sup>La utilizzeremo per poter simulare un ambiente Linux all'interno di una macchina Windows. Se si è già all'interno di un ambiente Linux, si potrà evitare l'utilizzo.

<sup>10</sup>Software open-source che permette di creare dei “contenitori” isolati, all'interno di un computer, dove verranno simulati dei sistemi operativi. Ne discuteremo meglio più avanti.

<sup>11</sup>Una Testnet è una blockchain alternativa utilizzata dai sviluppatori. I coin qui sono separati da quelli reali, possono essere generati e spesi simulando la blockchain reale.

<sup>12</sup>Bitcoin Core è un progetto open source, discendente dal client software originale di Bitcoin, costituito da due software “fullnode” per la validazione completa della blockchain e di un software per la gestione del portafoglio bitcoin.

<sup>13</sup>Per la precisione utilizzerò Ubuntu v20.4. A seconda della distribuzione utilizzata, potrebbero cambiare dei comandi o dei procedimenti.

### 3.4.1 Creazione di un'immagine su Docker

Docker è una Open Platform, con architettura client-server, per sviluppare, consegnare e lanciare applicazioni. Concettualmente simile ad una macchina virtuale, ma il suo funzionamento è totalmente differente da essa. Permette di contenere e lanciare un'applicazione, sotto forma d'immagine, in un ambiente semi-isolato chiamato *Container*.

Questi Container sono estremamente leggeri dato che non hanno bisogno, al contrario delle Macchine Virtuali, di un Hypervisor per poter funzionare dato che si poggiano direttamente sul kernel della macchina host. Le applicazioni vengono lanciate tramite delle immagini, esse sono dei modelli read-only con delle istruzioni, al loro interno, per creare un Container. Di conseguenza i Container sono delle istanze lanciabili di immagini. Essi possono essere creati, mossi o cancellati tramite il Docker API, possono anche essere interconnesse, tramite network, tra di loro.[6].

L'installazione di Docker ci permetterà di creare, più avanti, un'immagine di un sistema operativo Linux, per la precisione un sistema Alpine, con al suo interno la libreria “cert-issuer” e il software Bitcoin Core. Essendo il tutto svolto all'interno di un Container, l'utente che si approccia per le prime volte al progetto Blockcerts, potrà giocare tranquillamente all'interno di esso, senza doversi preoccupare dell'installazione di Bitcoin Core, di cert-issuer o di altri pacchetti specifici per il progetto. Inoltre i container vengono inizializzati sulla base delle immagini di partenza, ciò permette all'utente di “resettare” la macchina nel suo stato iniziale semplicemente chiudendo il container<sup>14</sup>.

Con queste premesse, consiglio l'utilizzo di questo tutorial installazione di Docker,

---

<sup>14</sup>Docker permette anche di “salvare” lo stato di un container tramite la creazione di un'immagine. In questa maniera sarà possibile salvare eventuali progressi o modifiche che si vogliono mantenere nel tempo.

con annessa firma dei certificati di prova, a chiunque si stia approcciando per la prima volta al mondo di Blockcerts e non è in possesso di una macchina con Bitcoin Core installato al suo interno; o che, semplicemente, non vuole passare per l'installazione del software di Bitcoin.

Per chi, invece, ha già scaricato Bitcoin Core o che ha una conoscenza minima di Blockcerts e del client bitcoin, consiglio di leggere questa sezione dato che la ritengo molto interessante, ma di non proseguire con l'installazione di Docker dato che, nel Capitolo 4, affronteremo passo passo la creazione personalizzata di un certificato, con annessa firma dello stesso, all'interno della Testnet, per poi proseguire con la Mainnet.

Passiamo ora alla sua installazione<sup>15</sup>. Creeremo delle repositories e procederemo l'installazione da lì. Andiamo sulla nostra home e premiamo la combinazione di tasti CTRL+ALT+T per poter aprire un terminale, da lì aggiorniamo *apt-get*, installando poi i pacchetti che ci permetteranno di utilizzare la repository sull'HTTPS.

```
$ sudo apt-get update

$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

Figura 3.5: Comandi apt package

Aggiungiamo la key GPG<sup>16</sup> ufficiale di Docker tramite il comando seguente.

---

<sup>15</sup>I passi di questa installazione possono essere approfonditi sulla pagina di GitHub della libreria cert-issuer e su Docker Docs

<sup>16</sup>GNU Privacy Guard è un software libero che permette di cifrare dei messaggi utilizzando un coppia di chiavi, pubblica e privata, generate dall'utente.



```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Figura 3.6: Comando curl per GPG

Per controllare che l'installazione e l'aggiunta della key siano andati a buon fine, possiamo utilizzare il prossimo comando per cercare la fingerprint tramite i suoi ultimi 8 caratteri.

```
$ sudo apt-key fingerprint 0EBFCD88

pub  rsa4096 2017-02-22 [SCEA]
     9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid      [ unknown] Docker Release (CE deb) <docker@docker.com>
sub  rsa4096 2017-02-22 [S]
```

Figura 3.7: Comandi controllo fingerprint

Per finire, utilizziamo il seguente comando per impostare la repository stabile di Docker.

```
$ sudo add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"
```

Figura 3.8: Comandi installazione repository stabile

Installiamo ora Docker. Prima di tutto aggiorniamo il pacchetto *apt-get* e installiamo poi l'ultima versione di *Docker Engine* e di *Containerd* disponibile.

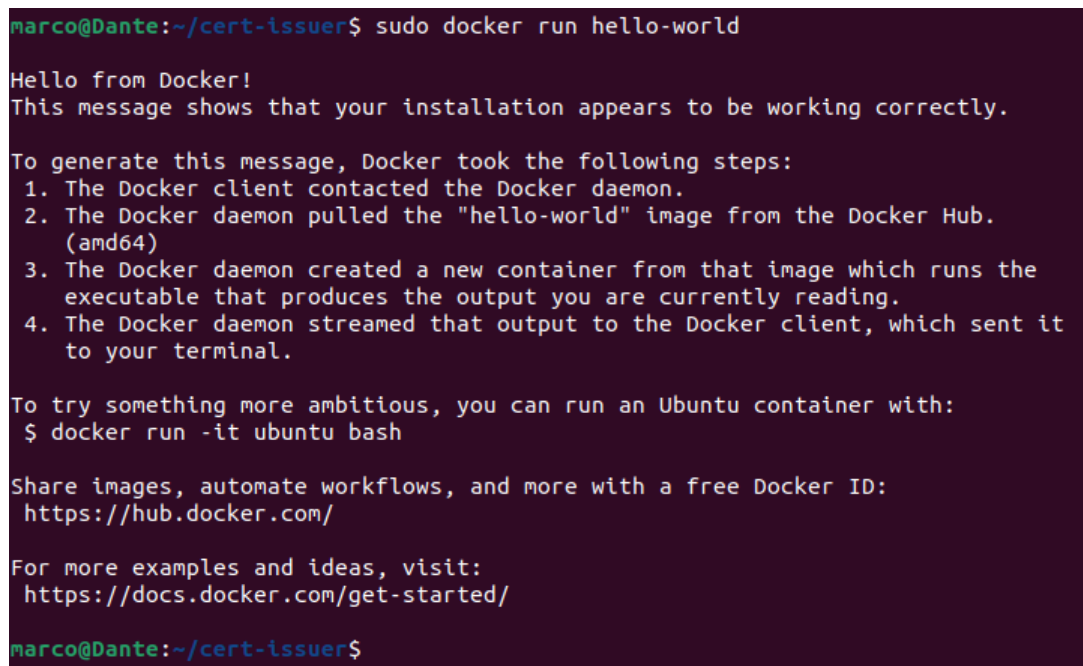
```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Figura 3.9: Comandi Installazione Docker

Concludiamo così l'installazione di Docker. Per controllare che sia tutto apposto, possiamo lanciare l'immagine *hello-world*. Immagine che verrà scaricata da Docker

(se non presente nel disco locale) e che verrà lanciata all'interno di un container.

Quando il contenitore si avvierà, farà vedere dei messaggi informativi e si chiuderà.



```
marco@Dante:~/cert-issuer$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

marco@Dante:~/cert-issuer$
```

Figura 3.10: L'immagine hello-world

### 3.4.2 Libreria cert-issuer

Come descritto nella sezione 3.3, questa libreria verrà utilizzata per firmare i certificati, da inserire all'interno della blockchain, tramite una transazione, dall'istituto emittente al beneficiario, che include l'hash del certificato stesso.

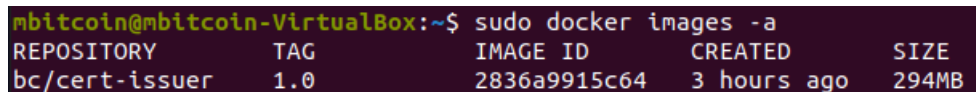
Per poterla utilizzare, cloniamo la repository su GitHub che contiene la libreria.

```
git clone https://github.com/blockchain-certificates/cert-issuer.git
```

Successivamente ci muoviamo all'interno della cartella appena clonata tramite il comando `cd` e creiamo, tramite il software di Docker precedentemente installato, la nostra immagine passando, come URL, la posizione corrente. Al tutto assegneremo, tramite il comando di opzioni `-t`, il tag "1.0".

```
cert-issuer && docker build -t bc/cert-issuer:1.0 .
```

Abbiamo così creato localmente un'immagine Linux che contiene la libreria *cert-issuer* e un client Bitcoin Core. Possiamo verificare la sua creazione tramite il seguente comando.



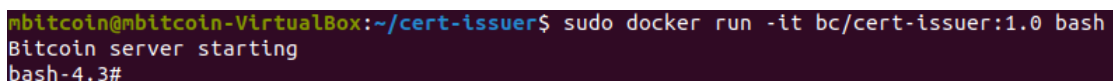
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
bc/cert-issuer	1.0	2836a9915c64	3 hours ago	294MB

Figura 3.11: Immagine di bc/cert-issuer con tag 1.0

Procediamo ora a lanciare l'immagine dentro un container tramite Docker. Utilizzeremo l'opzione l'opzione -it per poter creare un ambiente interattivo tramite shell. Ciò ci servirà a poter effettuare delle azioni all'interno del container stesso.

```
docker run -it bc/cert-issuer:1.0 bash
```

All'avvio del container, si avvierà anche il client di bitcoin in modalità regtest. Successivamente ci si aprirà un bash, sul terminale, che ci permetterà di eseguire delle operazioni all'interno del Container.



```
Bitcoin server starting
bash-4.3#
```

Figura 3.12: Avvio del container con il proprio bash.

All'interno di questo container eseguiremo la firma del certificato in una blockchain in modalità *regtests*. Come abbiamo visto, bitcoind si avvierà in automatico all'avvio della macchina, quindi passiamo subito alla creazione di un address che ci servirà per ricoprire il ruolo dell'issuer in questo test. Assegneremo il valore dell'address creato alla variabile "issuer" per semplificare la scrittura del codice ed evitare errori di trascrittura durante i prossimi comandi.

```
issuer="bitcoin-cli getnewaddress"
```

Salviamo ora l'indirizzo dell'issuer all'interno del file di configurazione "conf.ini".

File che useremo per gestire, successivamente, le firme dei certificati da emanare nella blockchain regtest.

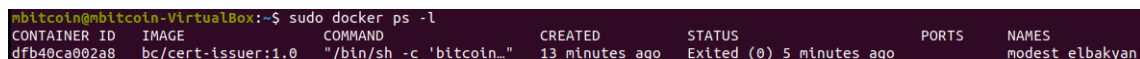
```
sed -i.bak "s/<issuing-address>/$issuer/g" /etc/cert-issuer/conf.ini
```

Infine, salviamo la Secret Key dell'address appena creato all'interno di un file di testo "pk\_issuer". Questo ci permetterà di accedere alla secret key in qualsiasi momento, facendo semplicemente riferimento al file di testo<sup>17</sup>.

```
bitcoin-cli dumpprivkey $issuer > /etc/cert-issuer/pk_issuer.txt
```

Abbiamo così creato e salvato il nostro indirizzo che useremo per impersonificare la figura dell'issuer all'interno del nostro test. Passiamo, infine, alla firma dei certificati.

Per il test che stiamo effettuando useremo dei modelli di certificati che si trovano già all'interno della libreria di cert-issuer del container. Se avessimo voluto utilizzare dei certificati creati da noi, fuori dal container stesso, avremmo dovuto fare riferimento all>ID del container, in cui l'immagine è stata lanciata, e copiarne il certificato all'interno.



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
dfb40ca002a8	bc/cert-issuer:1.0	"/bin/sh -c 'bitcoin..."	13 minutes ago	Exited (0) 5 minutes ago		modest_elbakyan

Figura 3.13: Comando per l>ID del contenitore dell'immagine

Comando per copiare il certificato esterno all'interno del container:

<sup>17</sup>Essendo un indirizzo creato all'interno di una regtest, che non verrà usato per mantenere dei BTC o per effettuare transazioni nel mondo reale, non c'è problema nel salvarsi in un file di testo libero la secret key. Sconsiglio caldamente, però, di fare la stessa cosa con un indirizzo bitcoin sensibile.

```
cp <cert-issuer-home>/data/unsigned_certificates/<your-cert-guid>.json
/etc/cert-issuer/data/unsigned_certificates/
```

I comandi sopra ci permetteranno di utilizzare Docker anche per firmare certificati differenti da quelli usati in questo test, ma ai fini dello stesso non proporremo degli esempi concreti.

Passiamo quindi alla firma dei modelli di certificati proposti dalla libreria stessa. Questi campioni di certificati si trovano fuori dall'ambiente di lavoro<sup>18</sup>, dobbiamo quindi copiarli all'interno di esso tramite il comando *cp*. Dato che entrambe le destinazioni si trovano all'interno del container, ci basterà effettuare un classico comando di copia passandogli degli indirizzi assoluti.

```
cp /cert-issuer/examples/
data-testnet/unsigned_certificates/3bc1a96a-3501-46ed-8f75-49612bbac257.json
/etc/cert-issuer/data/unsigned_certificates/
```

Dopo aver copiato il certificato all'interno dell'ambiente di lavoro, passiamo a gestire il nostro wallet per predisporre la generazione della transazione per l'invio del certificato firmato all'interno della blockchain regtest.

Il container inizia il suo stato con una blockchain vuota, senza possedere BTC ma con un wallet già caricato. Dato che stiamo usando la blockchain in modalità regtests, possiamo creare dei blocchi da minare tramite il seguente comando.

```
bitcoin-cli generate 101 && bitcoin-cli getbalance
```

---

<sup>18</sup>L'ambiente, come vedremo successivamente nel Capitolo 4, è definito all'interno del file "conf.ini" dentro la libreria cert-issuer. Si potrà modificare a proprio piacimento, ma per il test lavoreremo con la directory di default.

Il comando *bitcoin-cli generate 101* genererà 101 blocchi che verranno minati istantaneamente, da essi prenderemo delle fee. Mentre *bitcoin-cli getbalance* ci permette di vedere il quantitativo di BTC associati al nostro bitcoin address.

Adesso creeremo una transazione indirizzata all'issuing address creato nei passi precedenti. Il seguente comando invierà 5 BTC all'indirizzo assegnato, prendendoli dal nostro portafoglio. Ci risponderà con ID della transazione.

```
bitcoin-cli sendtoaddress $issuer 5
```

Questo passaggio ci assicurerà che, all'interno del nostro address, ci saranno i fondi necessari per effettuare il passo successivo.

Ora, tramite la libreria *cert-issuer*, firmiamo il certificato nella blockchain, ci basterà semplicemente richiamare questo comando, indirizzandolo al file di configurazione.

```
cert-issuer -c /etc/cert-issuer/conf.ini
```

Se tutto sarà andato a buon fine, troveremo il certificato appena firmato all'interno della directory “/etc/cert-issuer/data/blockchain\_certificates”. Questo certificato sarà in formato JSON e potrà essere visionato attraverso il sito di Blockcerts. Ci basterà collegarci, scegliere l'opzione del certificato JSON e trascinarlo al suo interno.

## Capitolo 4

# Personalizzazione dei certificati: la libreria cert-tools

Abbiamo appena concluso il percorso per poter firmare un certificato campione utilizzando il software Docker e una blockchain in modalità regtest. Se volessimo creare dei certificati personalizzati, potremmo farlo tramite il supporto della libreria “cert-tools”.

Essa ci permette di creare dei certificati, da firmare, utilizzando dati e immagini da noi fornite. La parte di libreria che useremo farà riferimento alla versione V2 di Blockcerts<sup>1</sup>. Essa creerà i nostri certificati facendo riferimento allo standard Open Badge<sup>2</sup>, infatti Blockcerts, nella sua versione V2, non è altro che un'estensione di questo standard applicato alla blockchain.

Per firmare questi certificati passeremo per la testnet di bitcoin, non più per la regtest, e per il software Bitcoin Core, non più tramite Docker. Il software dovrà essere presente, con la testnet scaricata, e funzionante all'interno della macchina dove creeremo e firmeremo i certificati.

---

<sup>1</sup>Attualmente è in sviluppo la versione V3 di Blockcerts, anche se la libreria presenta delle versioni alpha di alcuni comandi, rimaniamo sulla versione V2.

<sup>2</sup>Open Badges è un formato internazionale per la formattazione dei badge digitali. Per approfondire l'argomento, il loro sito ufficiale “Open Badges” fornisce tutte le informazioni necessarie.

In questa tesi non discuterò dell'installazione corretta di un full node bitcoin, di conseguenza daremo per scontata l'installazione e il funzionamento corretto del client di bitcoin. Nel caso in cui il lettore non sia a conoscenza dei procedimenti per l'installazione del software Bitcoin Core o sia totalmente nuovo al concetto, consiglio la lettura del libro [1].

In ogni caso, prima di procedere con l'installazione e la creazione dei certificati, se non si è in possesso di un wallet all'interno della testnet con dei fondi, consiglio di crearne uno prima di procedere con la lettura e utilizzare uno dei tanti servizi di faucet testnet per poter ottenere dei BTC da poter utilizzare sulla testnet. Passiamo ora alla creazione dei certificati.

Per poter creare un certificato personale tramite l'ausilio di Blockcerts dobbiamo, come prima cosa, scaricare la libreria in questione. Ci affideremo a GitHub per il download.

```
git clone https://github.com/blockchain-certificates/cert-tools.git && cd cert-tools
```

Tramite il comando `cd` ci siamo mossi all'interno della cartella appena scaricata. Da qui installeremo, tramite `pip` gli scripts che saranno necessari al nostro fine.

```
pip install .
```

Gli scripts appena installati sono “create-certificate-template” e “instantiate-certificate-batch”. Essi fanno riferimento, rispettivamente, ai comandi python `create_v2_certificate_template.py` e `instantiate_v2_certificate_batch.py`. Accettano in input un file di configurazione per la gestione della creazione del certificato, tratteremo meglio questo aspetto più avanti. Questi due file python si trovano all'interno directory “`./cert-tools/`”.

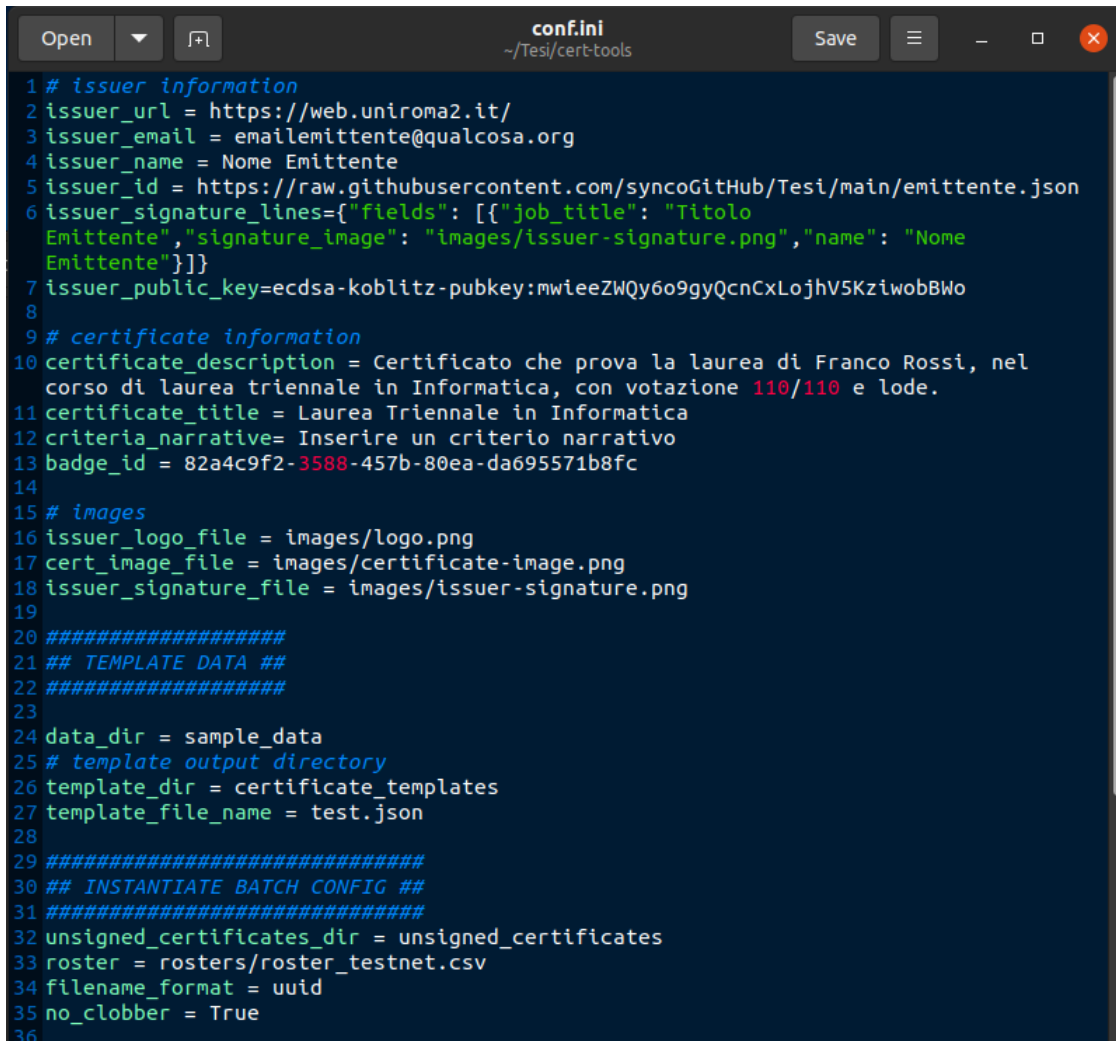


## 4.1 Configurazione dell'issuer ID

Prima di creare un template dei nostri certificati, dobbiamo prima definire i dati dell'issuer. Per far ciò utilizzeremo il programma python “create\_v2\_issuer.py” e gli passeremo, in input, il nostro file di configurazione. Passiamo quindi alla sua progettazione. All'interno della cartella cert-tools appena scaricata, troveremo il file “conf.ini”. Questo file dovrà essere compilato con i dati personali dell'issuer<sup>3</sup> prima di poter avviare il programma python. Per questo esempio, ho compilato il file in questione come nella immagine seguente.

---

<sup>3</sup>Inizialmente, per il lancio dell'applicazione python, i campi “issuer\_id” e “issuer\_revocation\_list” potranno essere lasciati vuoti, o comunque si potrà inserire l'URL della posizione in cui si hosteranno i file che ci apprestiamo a generare.



```

1 # issuer information
2 issuer_url = https://web.uniroma2.it/
3 issuer_email = emailmittente@qualcosa.org
4 issuer_name = Nome Emittente
5 issuer_id = https://raw.githubusercontent.com/syncoGitHub/Tesi/main/emittente.json
6 issuer_signature_lines=[{"fields": [{"job_title": "Titolo
Emittente", "signature_image": "images/issuer-signature.png", "name": "Nome
Emittente"}]}]
7 issuer_public_key=ecdsa-koblitz-pubkey:mwieeZWQy6o9gyQcnCxLojhV5KziwobBWo
8
9 # certificate information
10 certificate_description = Certificato che prova la laurea di Franco Rossi, nel
corso di laurea triennale in Informatica, con votazione 110/110 e lode.
11 certificate_title = Laurea Triennale in Informatica
12 criteria_narrative= Inserire un criterio narrativo
13 badge_id = 82a4c9f2-3588-457b-80ea-da695571b8fc
14
15 # images
16 issuer_logo_file = images/logo.png
17 cert_image_file = images/certificate-image.png
18 issuer_signature_file = images/issuer-signature.png
19
20 #####
21 ## TEMPLATE DATA ##
22 #####
23
24 data_dir = sample_data
25 # template output directory
26 template_dir = certificate_templates
27 template_file_name = test.json
28
29 #####
30 ## INSTANTIATE BATCH CONFIG ##
31 #####
32 unsigned_certificates_dir = unsigned_certificates
33 roster = rosters/roster_testnet.csv
34 filename_format = uuid
35 no_clobber = True
36

```

Figura 4.1: Configurazione file conf.ini della libreria cert-tools

Alcuni dei campi all'interno del file non sono ancora stati trattati ma lo faremo più avanti. Per ora possiamo notare che, come campo del “issuer\_id” abbiamo inserito un file, all'interno di una repository GitHub, ancora vuoto. Dopo la creazione del file con i dati personali del mittente, creerò una repository apposita che combaccerà con l'URL utilizzato in questo campo. Per questo scopo è possibile utilizzare qualsiasi servizio di hosting, compreso un local host sulla macchina. Ho deciso di

utilizzare proprio GitHub dato che ho intenzione di utilizzarlo anche in futuro<sup>4</sup>, trovo inoltre più facile per lo scopo di questa presentazione utilizzare una repository online piuttosto della creazione di un host locale.

Ora che abbiamo compilato il nostro file di configurazione, proseguiamo con la creazione del file “emittente.json”. Questo file conterrà le informazioni personali dell'emittente e dovrà essere sempre raggiungibile da chiunque voglia verificare, successivamente, il certificato da esso firmato. Per creare questo file, all'interno della cartella *cert-tools*, apriamo un terminale ed eseguiamo il seguente comando.

```
python3 cert_tools/create_v2_issuer.py -c conf.ini -o emittente.json -r None
```

Con questo comando stiamo eseguendo il file python, all'interno della directory “./cert-tools/” che ci permette di creare il file JSON. A questo programma stiamo passando le opzioni:

- *-c* per identificare il file di configurazione a cui dovrà fare riferimento durante la creazione del JSON.
- *-o* per decidere il nome del file finale.
- *-r* per specificare che non è presente nessuna lista di revocazione per i certificati che vogliamo creare.

Il file “emittente.json” verrà salvato all'interno della directory corrente. Ora dovremo inserirlo online nell'URL scelto inizialmente nel file “conf.ini”. Il file generato da questo procedimento sarà disponibile all'interno della seguente repository GitHub.

---

<sup>4</sup>La repository su GitHub rimarrà aperta a tutti anche dopo la scrittura di questa Tesi, inoltre caricherò alcuni file, che descriverò più avanti, all'interno di essa, aggiornandola con il tempo. In questa maniera spero di poter rendere più facile l'approccio a questa tecnologia per chi volesse approfondirla.

## 4.2 Creazione dei certificati non firmati

Ora che abbiamo creato, e messo online, il file json con le informazioni dell'emittente, passiamo alla creazione del template, con il quale firmeremo i certificati, e alla sua popolazione, con il CSV del roster, tramite l'ausilio degli script che abbiamo installato in precedenza all'inizio del Capitolo 4

### 4.2.1 Creazione del template

Questo template non conterrà dati specifici ma dei tag. Per poter creare questo file, chiamiamo il seguente script passandogli, tramite l'opzione `-c`, il file di configurazione "conf.ini" che abbiamo personalizzato.

```
create-certificate-template -c conf.ini
```

Esso creerà, all'interno della directory indicata dentro il file di configurazione, un file chiamato `test.json`. Per poter utilizzare questo template dobbiamo prima specificare chi sono i beneficiari.

### 4.2.2 Creazione delle istanze

All'interno di `cert-tools`, dentro la directory `data`, possiamo trovare una cartella contenente un file chiamato "roaster\_testnet.csv"<sup>5</sup>. Al suo interno dovremmo indicare tutti i beneficiari che dovranno ricevere un certificato inserendone i dati tramite la formattazione "Full Name", "Public Address" e "Email".

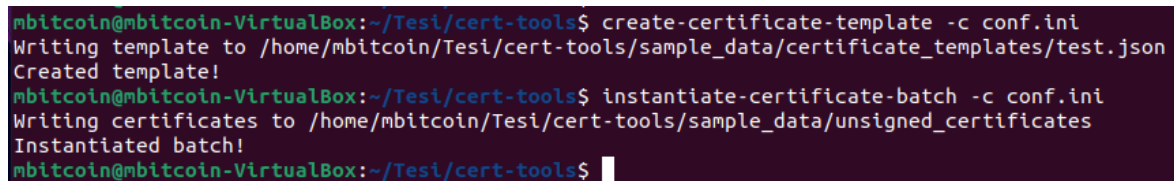
Dopo aver compilato il file csv, possiamo richiamare lo script seguente.

```
instantiate-certificate-batch -c conf.ini
```

---

<sup>5</sup>I file csv sono dei file con una determinata formattazione. Per la precisione, all'interno di questo file i dati sono divisi in righe e ogni dato è separato dal successivo tramite una virgola. Quando si finisce di inserire dei dati all'interno di una riga, si va a capo e si ricomincia il processo.

Esso popolerà il template precedentemente creato con i dati dei beneficiari all'interno del file CSV indicato all'interno del file di configurazione "conf.ini". Per ogni entry all'interno del roster verrà creato un certificato apposito all'interno della directory indicata nella configurazione.

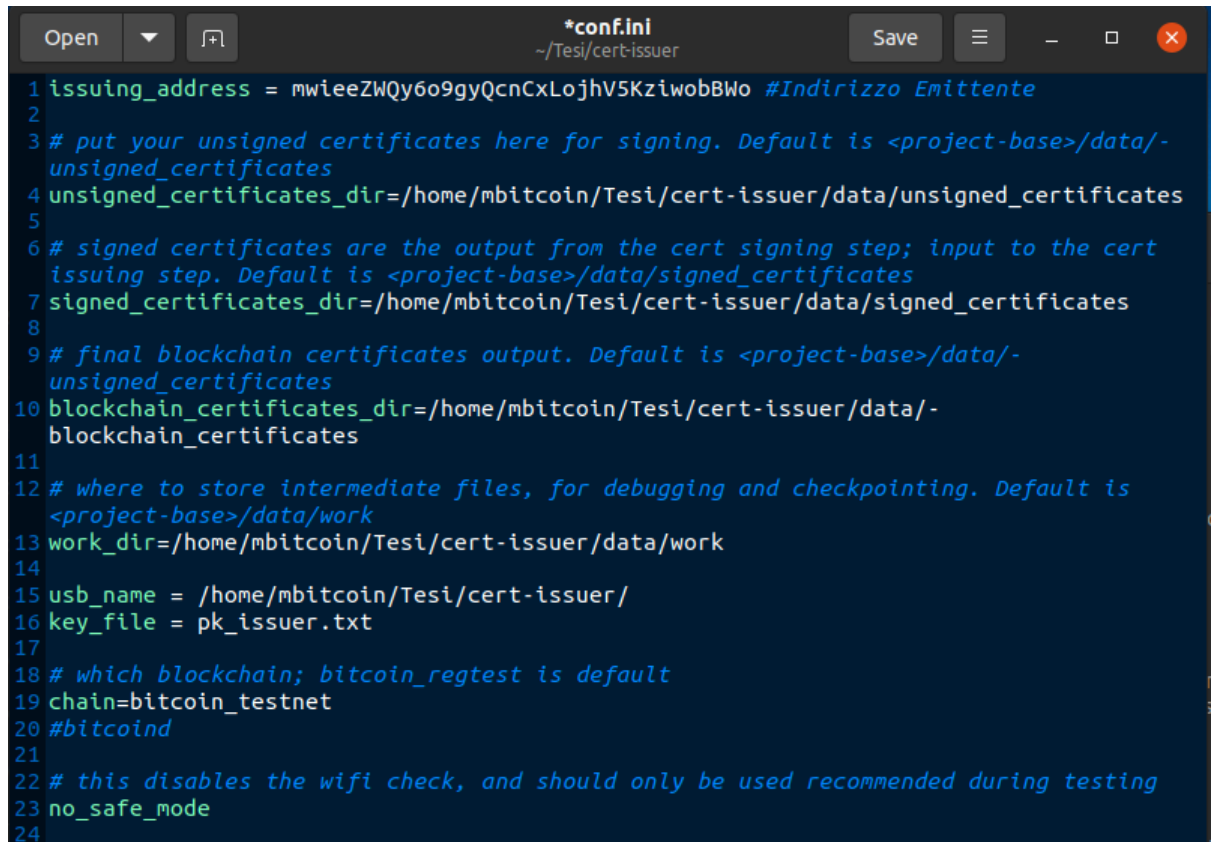


```
mbitcoin@mbitcoin-VirtualBox:~/Tesi/cert-tools$ create-certificate-template -c conf.ini
Writing template to /home/mbitcoin/Tesi/cert-tools/sample_data/certificate_templates/test.json
Created template!
mbitcoin@mbitcoin-VirtualBox:~/Tesi/cert-tools$ instantiate-certificate-batch -c conf.ini
Writing certificates to /home/mbitcoin/Tesi/cert-tools/sample_data/unsigned_certificates
Instantiated batch!
mbitcoin@mbitcoin-VirtualBox:~/Tesi/cert-tools$
```

Figura 4.2: Output degli script

## 4.3 Firma dei certificati all'interno della testnet di Bitcoin

Ora che abbiamo creato i certificati da firmare, non dobbiamo fare altro che firmarli tramite l'ausilio della libreria "cert-issuer" ed inserirli all'interno della testnet. Per poter procedere è necessario possedere abbastanza fondi all'interno dell'indirizzo associato all'emittente. Al momento della scrittura della tesi, le fee per poter effettuare la firma dei certificati sono di 133550 satoshi. Nel caso in cui non si possedessero abbastanza fondi, è possibile ricorrere all'ausilio delle faucet di bitcoin. Prima di procedere con la firma, dobbiamo compilare il file di configurazione di cert-issuer inserendo i dati del nostro ambiente di lavoro. Dobbiamo inoltre inserire l'address del nostro emittente, insieme al file dove abbiamo custodito la Secret Key corrispondente.



```

1 issuing_address = mwieeZWQy6o9gyQcnCxLojhV5Kziwob8Wo #Indirizzo Emittente
2
3 # put your unsigned certificates here for signing. Default is <project-base>/data/-
  unsigned_certificates
4 unsigned_certificates_dir=/home/mbitcoin/Tesi/cert-issuer/data/unsigned_certificates
5
6 # signed certificates are the output from the cert signing step; input to the cert
  issuing step. Default is <project-base>/data/signed_certificates
7 signed_certificates_dir=/home/mbitcoin/Tesi/cert-issuer/data/signed_certificates
8
9 # final blockchain certificates output. Default is <project-base>/data/-
  unsigned_certificates
10 blockchain_certificates_dir=/home/mbitcoin/Tesi/cert-issuer/data/-
  blockchain_certificates
11
12 # where to store intermediate files, for debugging and checkpointing. Default is
  <project-base>/data/work
13 work_dir=/home/mbitcoin/Tesi/cert-issuer/data/work
14
15 usb_name = /home/mbitcoin/Tesi/cert-issuer/
16 key_file = pk_issuer.txt
17
18 # which blockchain; bitcoin_regtest is default
19 chain=bitcoin_testnet
20 #bitcoind
21
22 # this disables the wifi check, and should only be used recommended during testing
23 no_safe_mode
24
  
```

Figura 4.3: Esempio di configurazione del file conf.ini

Infine, dobbiamo copiare i certificati, appena creati, all'interno della directory indicata nel file “conf.ini” della libreria cert-issuer. Possiamo quindi avviare un terminale e recarci all'interno della cartella cert-issuer ed eseguire il seguente comando che concluderà il nostro percorso con la firma dei nostri certificati e il loro inserimento all'interno della testnet.

```
cert-issuer -c conf.ini
```

Questo comando necessita del software *bicoind* per poter funzionare, motivo per cui esso deve essere, necessariamente, attivo in background prima di avviare il processo di firma. Dovremmo inoltre assicurarci che il wallet, con l'address dell'emittente, sia caricato all'interno del bitcoin-client.

Dopo aver effettuato i vari controlli, possiamo lanciare il programma.

```
mbitcoin@mbitcoin-VirtualBox: ~/Tesi/cert-issuer$ cert-issuer -c conf.ini
WARNING - Your app is configured to skip the wifi check when the USB is plugged in. Read the documentation to ensure this is what you want, since this is less secure
INFO - This run will try to issue on the bitcoin_testnet chain
INFO - Set cost constants to recommended_tx_fee=0.000600,min_per_output=0.000028,satoshi_per_byte=250
INFO - Processing 2 certificates
INFO - Processing 2 certificates under work path=/home/mbitcoin/Tesi/cert-issuer/data/work
INFO - Total cost will be 133500 satoshis
INFO - Starting finalizable signer
WARNING - app is configured to skip the wifi check when the USB is plugged in. Read the documentation to ensure this is what you want, since this is less secure
INFO - Stopping finalizable signer
WARNING - app is configured to skip the wifi check when the USB is plugged in. Read the documentation to ensure this is what you want, since this is less secure
INFO - here is the op_return_code data: 1612f34ba2bea79b9cd0ac120200d324075b492792df787b39a90fe4f9c8146d
INFO - Unsigned hextx=0100000001e1ba74850d57d51f3cb097cc160a0658cea1982d436e203db6d38c1ef453a8cd000000000ffff
ffff02e0570e00000000001976a914b1b84609ca7a66fc99b6630bc6476c75e254ca4388ac000000000000000226a201612f34ba2bea79b9cd0ac120200d324075b492792df787b39a90fe4f9c8146d00000000
INFO - Preparing tx for signing
INFO - Starting finalizable signer
WARNING - app is configured to skip the wifi check when the USB is plugged in. Read the documentation to ensure this is what you want, since this is less secure
INFO - Stopping finalizable signer
WARNING - app is configured to skip the wifi check when the USB is plugged in. Read the documentation to ensure this is what you want, since this is less secure
INFO - The actual transaction size is 235 bytes
INFO - Signed hextx=0100000001e1ba74850d57d51f3cb097cc160a0658cea1982d436e203db6d38c1ef453a8cd00000000b48304
5022100a0a2f8e8eb8246c42ed84fe3a68499f1cdfc32a1b1b957db572820e2cf49085b022056233f0b769c1eda433a9a421dde4f0b5
45aef08bf03d68d43465136a31a03c012102c30cae316b575549aaefd84e9cbc0764499b04edbc0170a7efbca4a3b663e38fffffffff0
2e0570e000000000001976a914b1b84609ca7a66fc99b6630bc6476c75e254ca4388ac000000000000000226a201612f34ba2bea79b9cd0ac120200d324075b492792df787b39a90fe4f9c8146d00000000
INFO - Signed hextx=0100000001e1ba74850d57d51f3cb097cc160a0658cea1982d436e203db6d38c1ef453a8cd00000000b48304
5022100a0a2f8e8eb8246c42ed84fe3a68499f1cdfc32a1b1b957db572820e2cf49085b022056233f0b769c1eda433a9a421dde4f0b5
45aef08bf03d68d43465136a31a03c012102c30cae316b575549aaefd84e9cbc0764499b04edbc0170a7efbca4a3b663e38fffffffff0
2e0570e000000000001976a914b1b84609ca7a66fc99b6630bc6476c75e254ca4388ac000000000000000226a201612f34ba2bea79b9cd0ac120200d324075b492792df787b39a90fe4f9c8146d00000000
INFO - verifying op_return value for transaction
INFO - verified OP_RETURN
INFO - Broadcasting succeeded with method_provider=<bound method BlockcypherProvider.broadcast_tx of <cert_issuer.blockchain_handlers.bitcoin.connectors.BlockcypherProvider object at 0x7f1148e36be0>>, txid=3ccb3f5b1ad391dcf44921f814ee7efed667a3bcca8736a4f23f4fa771044af
INFO - Broadcasting succeeded with method_provider=<bound method BlockstreamBroadcaster.broadcast_tx of <cert_issuer.blockchain_handlers.bitcoin.connectors.BlockstreamBroadcaster object at 0x7f1148e36c10>>, txid=3ccb3f5b1ad391dcf44921f814ee7efed667a3bcca8736a4f23f4fa771044af
INFO - Broadcast transaction with txid 3ccb3f5b1ad391dcf44921f814ee7efed667a3bcca8736a4f23f4fa771044af
INFO - Your Blockchain Certificates are in /home/mbitcoin/Tesi/cert-issuer/data/blockchain_certificates
mbitcoin@mbitcoin-VirtualBox:~/Tesi/cert-issuer$
```

Figura 4.4: Output dello script cert-issuer sulla testnet

Se tutto è andato bene, troveremo i nostri certificati nella directory descritta nel file di configurazione. Questi certificati sono pronti e firmati, potranno essere tranquillamente verificati in pochi passaggi da chiunque possenga le credenziali necessarie.

## 4.4 Visualizzazione del Certificato tramite Blockcerts

Blockcerts offre una soluzione veloce ed efficace per la visione dei certificati direttamente nella sua home page. Ci basterà trascinare il file JSON all'interno della pagina web, proprio come abbiamo fatto per i certificati campione descritti nella Sezione 3.4.2

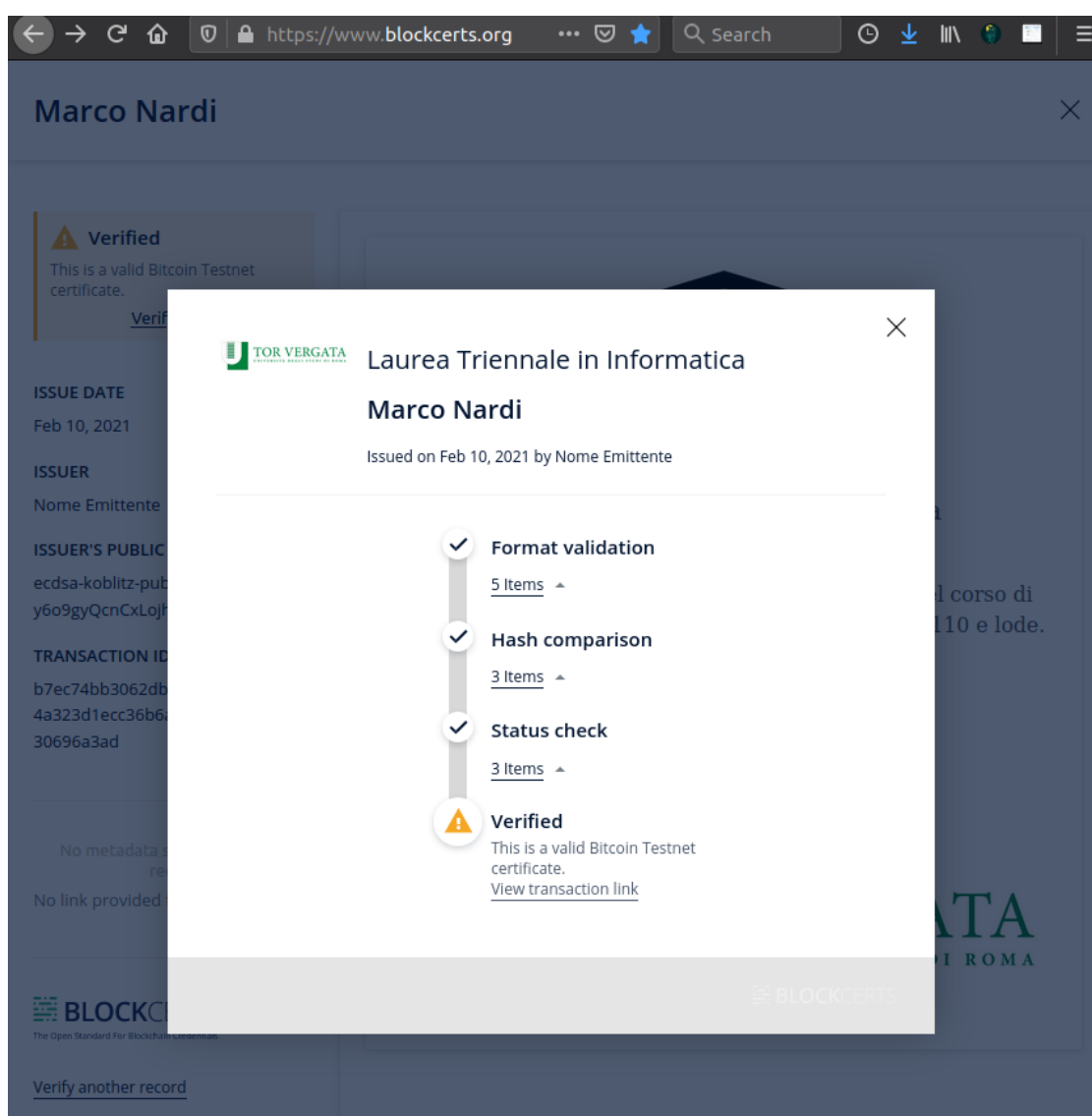


Figura 4.5: Verifica della validità del certificato



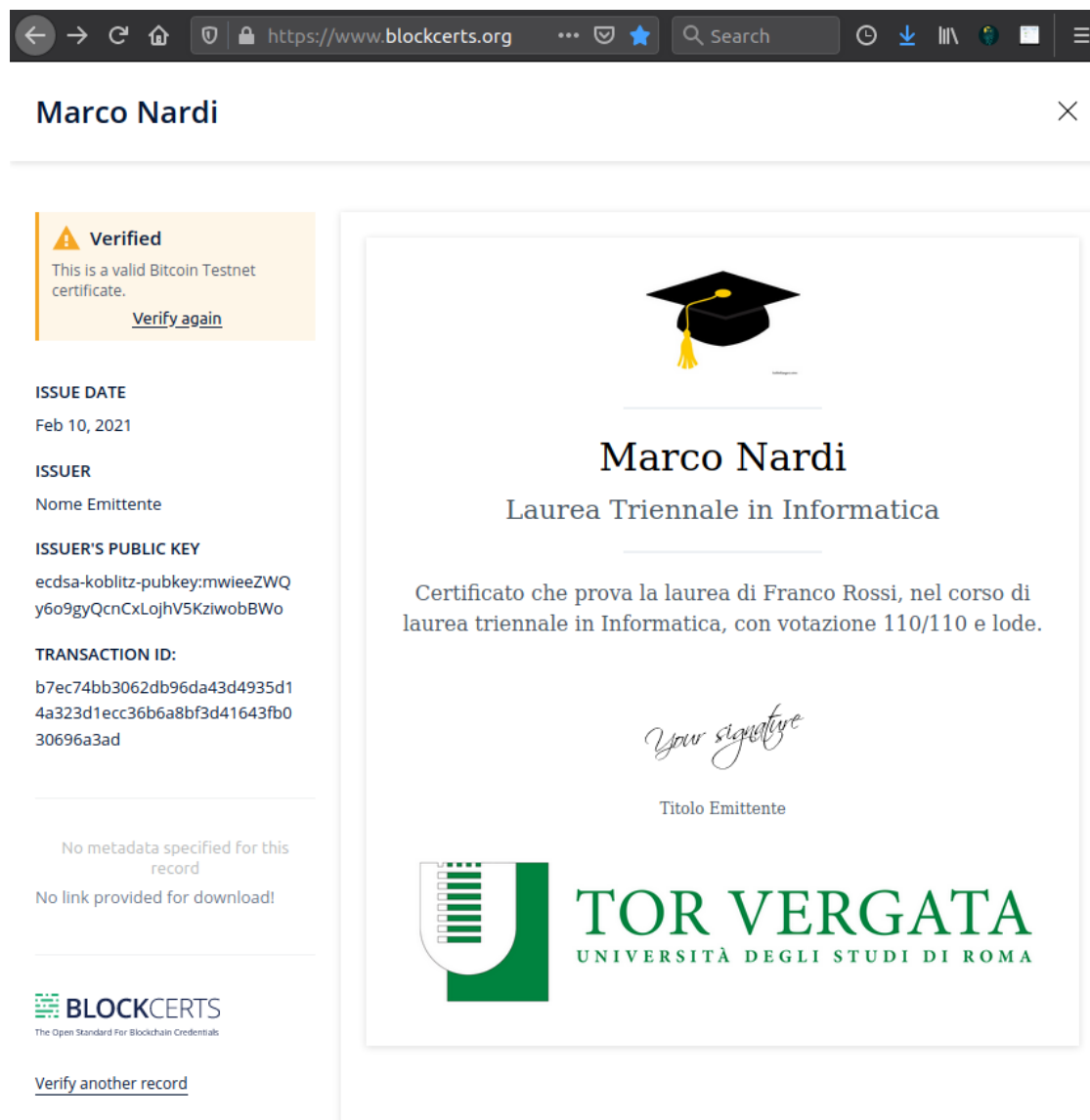


Figura 4.6: Presentazione finale del certificato

Il certificato che abbiamo appena costruito è estremamente semplice e molto basilare dal punto di vista della personalizzazione, ma tramite delle librerie fornite da Blockcerts è possibile modificare la presentazione del certificato, aggiungendo feature come link per il download del certificato stesso, visionare il certificato con un tema scuro e così via. Le possibilità di personalizzazione ci sono e sono molto vaste. Ho deciso di non trattarle per rendere questa sezione più facile per chi si appropria

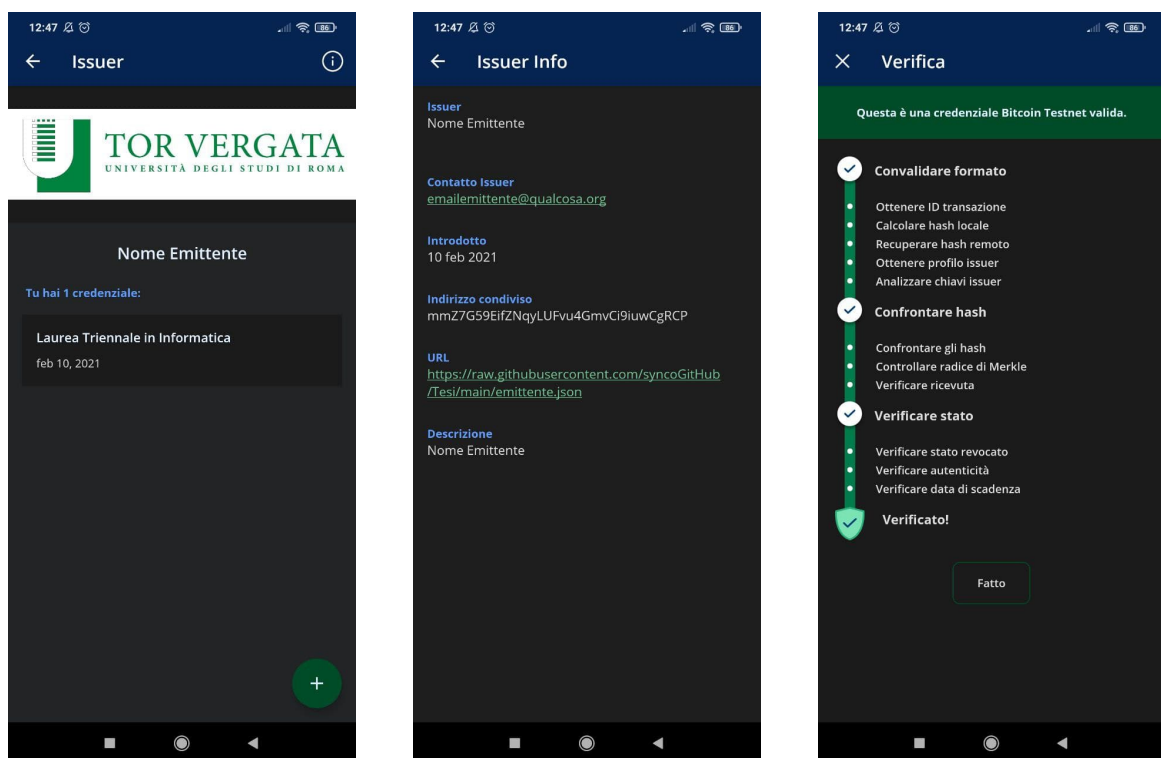
a questo mondo, ma la personalizzazione del certificato stesso sarebbe il prossimo passo per poter emanare dei certificati personalizzati.

## 4.5 Visualizzazione app Android

Blockcerts ha sviluppato anche un'applicazione Android che permette di portare con sé i certificati comodamente all'interno dello smartphone.

I passaggi per l'installazione e dell'inserimento dei certificati sono estremamente facili, inoltre l'applicazione permette, oltre che di visualizzare il certificato in qualsiasi momento, di verificare l'autenticità e la validità dello stesso in pochissimi secondi.

Inoltre controllerà se il certificato è stato revocato o meno.



(a) Presentazione del certificato all'interno dell'app android

(b) Informazioni dell'emittente del certificato

(c) Verifica delle credenziali tramite app android

Figura 4.7: Alcuni screen dell'applicazione Android di Blockcerts

## Capitolo 5

# Un ambiente user-friendly per Blockcerts

Il progetto Blockcerts è estremamente interessante ma, allo stesso tempo, molto difficile da approcciare. Nonostante ogni libreria avesse una pagina GitHub dedicata, la lettura e l'applicazione si sono rivelate molto complesse. Inoltre i codici e i file di configurazione all'interno delle varie librerie erano mischiati insieme alle versioni precedenti. Tutto ciò, insieme alle molte librerie esistenti e ad altre problematiche incontrate, ha reso il mio approccio a Blockcerts molto lento e macchinoso.

Tali difficoltà nuocciono alla salute del progetto stesso, poiché creano un muro per gli sviluppatori che vogliono approcciarvisi e per gli utenti che vorrebbero integrare Blockcerts nel loro sistema di emissione di certificati.

In questa sede si è quindi voluto creare un ambiente in Python che potesse semplificare il processo base di creazione dei certificati stessi. Si è quindi resa la creazione dei certificati personalizzati molto più user-friendly e adatta all'utilizzo concreto, poiché le figure che se ne occupano nel mondo reale non richiedono conoscenze nel campo di Bitcoin e del progetto Blockcerts stesso.

Per ottenere ciò mi sono basato sulle librerie precedentemente utilizzate nei Capitoli 3 e 4, progettando i miei programmi in maniera tale che si interfacciassero con esse,

senza modificarne il comportamento o l'essenza. In questa maniera non ho intaccato il progetto ma, allo stesso tempo, ho eliminato la sua parte macchinosa e l'ho reso compatibile e applicabile ad un ambiente universitario.

Passerò ora ad elencare le varie problematiche e difficoltà affrontate durante il progetto per poi passare alla presentazione del mio ambiente Python con degli esempi di casi d'uso.

## 5.1 Le problematiche incontrate

Durante il mio percorso con Blockcerts ho incontrato molte difficoltà, per la maggior parte dovuto a delle incompatibilità dei requisiti e pacchetti richiesti delle librerie stesse. Non è raro installare una loro libreria e incontrare dei conflitti di pacchetti dovuti ai requisiti imposti da essa. Di conseguenza ho dovuto spesso modificare i requisiti delle versioni richieste ai pacchetti.

Inoltre le librerie non sono aggiornate sempre alle ultime versioni, infatti esse possono essere considerate più delle vere e proprie repository GitHub dove gli sviluppatori del progetto caricano i vari aggiornamenti tramite dei “commit” di nuovi file, evitando di cancellare le vecchie versioni del programma stesso ma lasciandole come riferimento.

Ciò fa sì che alcune funzioni vengano tralasciate nei vari aggiornamenti e che non vengano descritte, ed analizzate, all'interno della documentazione della libreria.

Inoltre le funzioni che vengono aggiornate alla versione superiore, spesso, non vengono eliminate e ciò fa sì che ci siano più funzioni, con versioni differenti, che danno lo stesso output. Creando dei problemi d'incompatibilità nell'eventualità in cui si adoperino delle funzioni, di differente versione, durante la creazione del certificato. Tutti questi problemi delle librerie Open Source non rendono assolutamente il pro-

getto Blockcerts meno valido, ma rendono un possibile primo approccio di uno sviluppatore, o di un ente, interessato al progetto molto difficile e, a tratti, frustrante.

## 5.2 Un approccio laborioso

Attualmente per poter creare dei certificati personalizzati, cambiare ambiente di lavoro ed utilizzare un file CSV personale, è necessario navigare manualmente all'interno dei vari file “conf.ini” delle librerie cert-tools e cert-issuer. Ciò rende molto macchinoso il processo di modifica dei dati stessi, inoltre lo rende soggetto ad errori d'inserimento dato che la semplice cancellazione di un carattere all'interno dei file in questione farebbe fallire tutto il processo. Inoltre bisognerà interfacciarsi con il client di Bitcoin per poter caricare i wallet e gli indirizzi necessari per la firma dei certificati.

Queste operazioni rendono difficile e macchinosa la modifica e la creazione di certificati personalizzati, soprattutto se si parla di una grande mole di beneficiari con certificati differenti. Infatti, ad ogni piccola modifica all'interno dei parametri del certificato, sarà necessario procedere con la modifica dei file di configurazione, alla modifica del campo ID e ad una nuova creazione, e popolazione, del Template associato.

Questo processo non sarebbe un problema se il certificato da creare fosse uno solo, ma per l'utilizzo di questo standard all'interno di un'istituzione come un'università, esso diventa un problema reale che rende complicato il suo l'utilizzo.

## 5.3 Gestione del processo tramite Python

Quindi tutte queste difficoltà rendono il progetto Blockcerts complicato per chi non ha esperienze nel settore o per chi deve firmare più certificati differenti tra loro,

entrambi aspetti che possiamo ritrovare nell'utilizzo universitario.

In questo ambiente, la figura dell'issuer si troverà a dover gestire una mole importante di certificati differenti tra loro, inoltre questa figura non sarà necessariamente fissa, di conseguenza si dovranno cambiare spesso i file di configurazione. Inoltre il progetto Blockcerts unisce sotto un unico aspetto la firma dei certificati e la gestione del roster, cosa che nella realtà universitaria è ben differente. Infatti, qui il contenuto del file roster dovrebbe essere gestito dalla segreteria che non avrà necessariamente le competenze per poter gestire dei wallet Bitcoin, né le competenze per la creazione di vari indirizzi. Il mio progetto in Python punta a modificare questa scelta di Blockcerts, senza effettivamente andare ad intaccare il progetto stesso, tramite l'ausilio di due programmi indipendenti. Esso divide il progetto Blockcerts in due lati, uno amministrativo che punta a gestire la lista di utenti che devono ricevere un certificato firmato e l'altro dedicato all'issuer, alla raccolta delle sue info e della firma stessa dei certificati.

### **5.3.1 Lato Organizzativo: gestione del file roster**

Questo programma punta a modificare in file CVS, utilizzato per popolare il Template, in una maniera più dinamica e con un'interfaccia più user-friendly.

Il lato di compilazione della lista di beneficiari viene gestito dal programma "modificatoreRoster.py". Esso si occupa di presentare un'interfaccia di facile utilizzo dove l'utente possa compilare il file CSV in autonomia e senza una particolare conoscenza di Bitcoin.

È stato creato con l'idea di essere utilizzato dalla segreteria dell'università. In questo ambiente non sono richieste delle competenze inerenti al progetto Blockcerts, di conseguenza risulterebbe difficile, per l'utente finale, utilizzare il client Bitcoin per

fornire degli address a chi effettua una richiesta ma non ne è in possesso.

Con questo pensiero in mente, ho puntato ad un approccio più schematico e semplice, che permettesse a chiunque, anche a chi non ha conoscenza dello standard CSV e di Bitcoin, di poter compilare con facilità il file stesso. Inoltre sarà possibile tramite il pulsante “Address” aggiungere una riga alla tabella con, già, un indirizzo Bitcoin valido e pronto all’utilizzo. Tramite di esso, l’utente non dovrà creare di sua mano un address Bitcoin e non dovrà avere, necessariamente, delle conoscenze specifiche del capo.

In questa maniera il nuovo standard verrebbe accettato con molta più facilità e non andrebbe a gravare, tramite il suo utilizzo, sul personale, cosa che renderebbe difficile un’integrazione corretta e sana di esso.

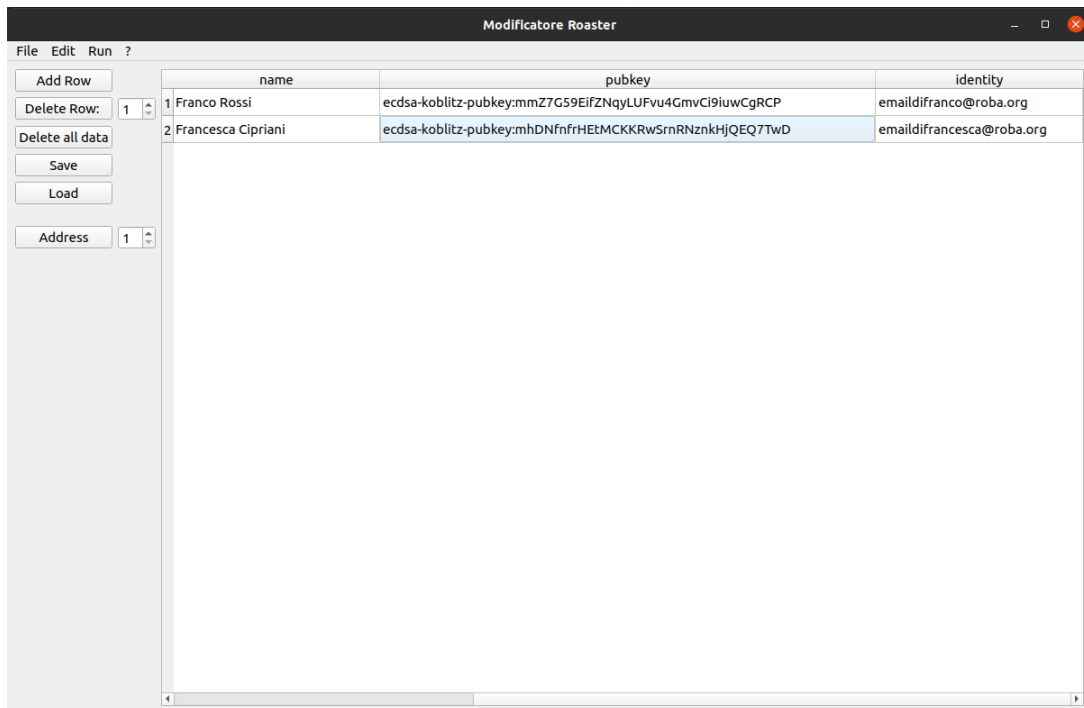


Figura 5.1: Schermata del programma per la gestione del file CSV

L’interfaccia del programma è stata resa il più possibile semplice e minimale, senza

sacrificare le sue funzionalità.

L'utente, tramite il pulsante "Add Row" potrà aggiungere delle righe vuote, alla tabella, che potranno essere modificate semplicemente cliccandoci e scrivendoci sopra.

I pulsanti "Delete Row" e "Delete all data" si occupano di cancellare, rispettivamente, solo la riga indicata all'interno della casellina alla sua destra e l'intera tabella.

I classici pulsanti "Save" e "Load" si occupano di salvare la tabella, sovrascrivendola al file CSV di destinazione, e di caricare i dati all'interno del file CSV direttamente alla tabella. Inoltre, la funzione di load, verrà avviata in automatico all'avvio del programma stesso.

Di default, il file che viene modificato si trova all'interno della libreria cert-tools, ma una copia verrà effettuata ogni volta che la funzione di salvataggio entra in azione.

La cartella di default, dove verrà inserita la copia, si trova nella stessa directory del programma, nella cartella "output".

### **5.3.2 Lato Issuer: inserimento dati e firma dei certificati**

Questo programma, invece, è stato creato per permettere l'inserimento delle credenziali dell'issuer in una maniera più fluida e semplice, senza dover andare a modificare i vari file di configurazione all'interno delle librerie di Blockcerts per ogni piccolo cambiamento.

I campi che vengono visualizzati corrispondono ai campi di configurazione all'interno dei vari file "conf.ini" delle librerie utilizzate. Mentre, il campo "CONSOLE DI OUTPUT" permette di visualizzare gli errori, i successi e ogni altra informazione che le funzioni delle librerie utilizzate forniscono. Quindi, tramite esso potremmo monitorare l'andamento della creazione, della firma e dei salvataggi dei nostri dati.



Nel momento in cui il programma viene avviato, esso crea un dizionario vuoto e una funzione che va a leggere i vari campi, all'interno dei file di configurazione, per inserirli all'interno del dizionario tramite una relazione chiave-valore. In questa maniera, l'applicazione potrà contare su un dizionario aggiornato dove ogni chiave corrisponde al campo richiesto da uno dei file "conf.ini", mentre i valori del dizionario corrisponderanno ai valori dei suddetti campi.

Tramite questo dizionario, sempre all'avvio del programma, inseriremo i vari valori dei file di configurazione all'interno delle loro box corrispondenti. In questa maniera l'utente avrà un visione completa dei parametri, già presenti, all'interno dei file di configurazione.

La funzione di salvataggio invece, avviata tramite il pulsante "Salva le impostazioni", prenderà i valori all'interno del nostro dizionario e li inserirà all'interno dei file di configurazione "conf.ini".

Configurazione dell'emittente

-- Informazioni necessarie alla firma del certificato all'interno della mainnet --

### Info dell'emittente

Nome completo :  Nome file con firma :  Link ID Online :

Titolo di prestigio :  Link alla pagina web :

Email di servizio :  Public Key :

Badge ID personale :  Password :

### Descrizione del certificato

Titolo del certificato :

Descrizione del certificato :

Narrazione :

CONSOLE DI OUTPUT  
Caricate le impostazioni in memoria ...

Figura 5.2: Schermata del programma per la gestione delle info dell'issuer

I quattro pulsanti “Crea file ID”, “Crea Template”, “Popola il Template” e “Firma i certificati” permettono di avviare le funzioni corrispondenti, della libreria di Blockcerts, e il box della console permetterà di monitorare i cambiamenti e gli output delle suddette funzioni. In questa maniera si è in controllo di tutto il processo di firma dei certificati tramite solo questa interfaccia. Infatti i vari file, creati tramite questi pulsanti, si troveranno già nella loro destinazione finale e l'utente non dovrà intervenire per spostarli tra un processo e l'altro. In ogni caso, sempre nella cartella “output” verranno creati delle copie dei file creati, che potranno essere visionati dall'utente per poter controllare possibili errori.

- “Crea file ID” chiamerà la funzione “create\_v2\_issuer.py” all'interno della libreria Blockcerts, passandogli il file di configurazione “conf.ini”, della libreria

cert-tools, e l'opzione "-r None".

Essa creerà il file di output, con nome di default "issuer.json", all'interno della cartella "output".

- "Crea Template" chiamerà la funzione "create-certificate-template" passandogli il file di configurazione "conf.ini", della librerie cert-tools.

Essa creerà il template, con il nome di default "test.json", e ne farà una copia da inserire all'interno della cartella "output".

- "Popola il Template" chiamerà la funzione "instantiate-certificate-batch" passandogli il file di configurazione "conf.ini", della librerie cert-tools.

Essa andrà a cercare, in una directory di default, il file roster.csv che dovrà essere fornito dal lato amministrativo, ovvero dagli utenti del programma precedente. Dopo aver trovato il file, esso lo utilizzerà per creare, tramite il template creato con "Crea Template", una serie di certificati che dovranno, poi, essere firmati. Ogni file da firmare verrà copiato all'interno della cartella "output".

- Infine, il pulsante "Firma i certificati", chiamerà la funzione "cert-issuer" passandogli il file di configurazione "conf.ini", della librerie cert-issuer.

Essa prenderà i certificati all'interno della libreria cert-issuer, caricherà il wallet Bitcoin di proprietà dell'università e utilizzerà la password inserita nel campo "Password" per aprire un file criptato, all'interno di una destinazione di default, che conterrà la Secret Key per poter firmare i certificati stessi. Successivamente proseguirà con la firma dei certificati all'interno della blockchain e restituirà, sempre all'interno della cartella "output", i certificati firmati e pronti per essere distribuiti.

Se durante questo procedimento la password fosse sbagliata, non sarebbe possibile firmare i certificati ed utilizzare i soldi all'interno del wallet. Questo permette di controllare chi ha accesso al wallet e chi, soprattutto, sia in possesso dei requisiti per usufruire dei soldi all'interno di quell'address Bitcoin.

L'unione di questi due programmi permette l'utilizzo di Blockcerts come un possibile standard per l'ambiente universitario di Tor Vergata, dividendo il peso del processo tra amministrazione e professori. Inoltre, non è necessario un corso specialistico per utilizzarli e comprenderli a fondo, di conseguenza non graverebbero sul sistema attuale se venissero inseriti come progetto in via di sviluppo.

Ovviamente i programmi sono ancora grezzi e possono essere migliorati ed adattati meglio per integrarsi, senza problemi, alla vita della nostra università. L'effort richiesto per effettuare questo adattamento, però, sarebbe sicuramente ricompensato con un sistema altamente tecnologico e funzionale per l'emissione di certificati su una tecnologia Blockchain.

## Capitolo 6

### Conclusioni

Con questo studio ho voluto analizzare la possibilità di utilizzare la libreria Open Source del progetto Blockcerts per come standard per la digitalizzazione dei certificati di Laurea. Tramite il mio progetto ho voluto presentare un'interfaccia GUI per rendere il processo di digitalizzazione user-friendly e accessibile ad un possibile affiancamento dell'attuale sistema di emissione di certificati.

In un mondo sempre più digitale penso che quello di digitalizzare i certificati tramite l'ausilio di una Blockchain sia un passo da dover compiere e il progetto Blockcerts può, tranquillamente, fungere da piano di appoggio iniziale per uno sviluppo futuro. Attualmente, gli atenei di Padova e Milano Bicocca forniscono questo servizio tramite lo standard Blockcerts, quindi l'università di Tor Vergata non si troverebbe da sola ad affrontare questa sfida. Inoltre i suoi studenti gioverebbero di tutti i lati positivi di avere un certificato digitale a portata di mano che li aiuterà a muoversi tra le varie università e nel mondo del lavoro. Per la mia ricerca ho utilizzato la versione V2 della libreria Blockcerts ma, al momento della scrittura, si sta già discutendo di una possibile versione V3 che dovrebbe utilizzare uno standard di verifica delle credenziali insieme ad una firma tramite la "Merkle Proof 2019". Essa non sarà retro-compatibile con la versione V2 di Blockcerts e, i certificati creati tramite lo standard

V2 non potranno essere firmati con la versione V3.

Ciò ha posto un limite alla mia tesi dato che non è ancora possibile utilizzare la versione V3 e, presto, la versione V2 potrebbe diventare obsoleta.

Per chi volesse proseguire la mia ricerca, Blockcerts fornisce una libreria Open Source per la modifica della visualizzazione dei certificati. Tramite di essa si potrebbe creare un visualizzatore di certificati personale che possa mostrare il certificato rispecchiando al meglio i bisogni dell'università di appartenenza.

Sarebbe anche possibile replicare il mio percorso con una blockchain differente da Bitcoin o di creare un sistema che possa permettere l'emissione di certificati compatibili con un documento di certificazione di Laurea con esami.

Inoltre, si potrebbe utilizzare la futura versione V3 dello standard Blockcerts o creare una propria libreria che possa replicare il funzionamento di Blockcerts ma che, allo stesso tempo, sia ottimizzata per l'ambiente di sviluppo per cui è stata pensata.

# Elenco delle figure

1.1	Esempi di Input e Output di una funzione hash criptografica . . . . .	10
1.2	Illustrazione Logica dell'Algoritmo di Criptografia Asimmetrico . . .	13
1.3	Struttura Logica Interna di una Transazione . . . . .	16
1.4	Struttura Logica di una Transazione . . . . .	17
1.5	Grafico del totale delle fee, in BTC, pagate ai Miner dal 2020 al 2021	19
1.6	Esempio di un Merkle Root . . . . .	22
1.7	Grafico dell'Hash Rate del giorno 14/01/2021 con prospettiva annuale	27
2.1	Esempio logico di blocchi collegati tra loro all'interno della blockchain	31
2.2	Esempio Logico di una Soft Fork . . . . .	34
2.3	Esempio Logico di una Hard Fork . . . . .	35
2.4	Hash Rate del 2016 –"bitcoinwisdom.com" . . . . .	39
3.1	Esempio logico di un caso d'uso [10] . . . . .	43
3.2	Schema logico dei passaggi[3] . . . . .	45
3.3	Transazione di Blockcerts per la firma dei certificati . . . . .	46
3.4	Esempio di costruzione di un merkle tree con l'hash dei certificati . .	47
3.5	Comandi apt package . . . . .	51
3.6	Comando curl per GPG . . . . .	52
3.7	Comandi controllo fingerprint . . . . .	52

3.8	Comandi installazione repository stabile . . . . .	52
3.9	Comandi Installazione Docker . . . . .	52
3.10	L'immagine hello-world . . . . .	53
3.11	Immagine di bc/cert-issuer con tag 1.0 . . . . .	54
3.12	Avvio del container con il proprio bash. . . . .	54
3.13	Comando per l'ID del contenitore dell'immagine . . . . .	55
4.1	Configurazione file conf.ini della libreria cert-tools . . . . .	61
4.2	Output degli script . . . . .	64
4.3	Esempio di configurazione del file conf.ini . . . . .	65
4.4	Output dello script cert-issuer sulla testnet . . . . .	66
4.5	Verifica della validità del certificato . . . . .	67
4.6	Presentazione finale del certificato . . . . .	68
4.7	Alcuni screen dell'applicazione Android di Blockcerts . . . . .	69
5.1	Schermata del programma per la gestione del file CSV . . . . .	74
5.2	Schermata del programma per la gestione delle info dell'issuer . . . . .	77



# Bibliografia

- [1] Andreas M. Antonopoulos. *Mastering Bitcoin - Programming the open blockchain*. United States of America: O'Reilly Media, Inc. 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2017.
- [2] *Bitcoin Core*. URL: <https://bitcoin.org/en/bitcoin-core/>.
- [3] *Blockcerts*. URL: <https://www.blockcerts.org/>.
- [4] Phil Champagne. *The Book Of Satoshi The Collected Writings of Bitcoin Creator Satoshi Nakamoto*. United States of America: e53 Publishing LLC, 2014.
- [5] *Coin Market Cap*. URL: <https://coinmarketcap.com/it/currencies/bitcoin/>.
- [6] *Docker*. URL: <https://www.docker.com/>.
- [7] Pedro Franco. *Understanding Bitcoin Cryptography, Engineering and Economics*. United Kingdom: TJ International Ltd, 2015.
- [8] Arvind Narayanan Joseph Bonneau Edward Felten Andrew Miller Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies*. United Kingdom: Princeton University Press, 2016.
- [9] David Hoppe. *Bitcoin's Centralization Problem*. 2018. URL: [gammalaw.com/bitcoins\\_centralization\\_problem/](http://gammalaw.com/bitcoins_centralization_problem/).
- [10] *Hyland*. URL: <https://www.hyland.com/en>.
- [11] Ari Juels Markus Jakobsson. *Proofs of Work and bread pudding protocols (extended abstract)*. 1999.
- [12] Iraj Sadegh Amiri Mohammad Reza Khalifeh Soltanian. *Theoretical and Experimental Methods for Defending Against DDoS Attacks*. Elsevier, 2016.
- [13] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2009.
- [14] Tadge Dryja Neha Narula. *Cryptocurrency Engineering and Design*. 2018. URL: <https://ocw.mit.edu/courses/media-arts-and-sciences/mas-s62-cryptocurrency-engineering-and-design-spring-2018/index.htm>.
- [15] *SHattered*. URL: <https://shattered.io/>.