Indice

Elenco delle figure

Capitolo 1

Introduzione

Introduzione

Capitolo 2

Origini di Blockchain

2.1 Il problema iniziale

Prima dell'avvento di Bitcoin era impossibile prescindere dalla mediazione di un sistema centrale per validare dei dati informatici sensibili quanto possono esserlo delle transazioni economiche.
Immaginiamo di dover creare del "valore digitale" adatto ad essere trasferito in maniera analoga a quanto viene fatto normalmente con il contante. Un primo ingenuo passaggio è quello di associare un valore ad un'entità digitale, sia essa una stringa, un numero, un file o un generico dato, liberamente accessibile oppure in codice. Per loro natura però, i dati digitali sono estremamente facili da replicare: trattandosi in fondo di sequenze di 0 e 1 più o meno lunghe, generarne una copia esatta è un procedimento agevole e sostanzialmente privo di costo. Questo si scontra fortemente con il proposito di creare del valore, dal momento che si ha la necessità di rendere *scarso* il bene. Prendiamo ad esempio il denaro contante: non è accettabile che la stessa cifra sia trasferita due volte dalla stessa persona a due destinatari differenti semplicemente replicando la banconota.

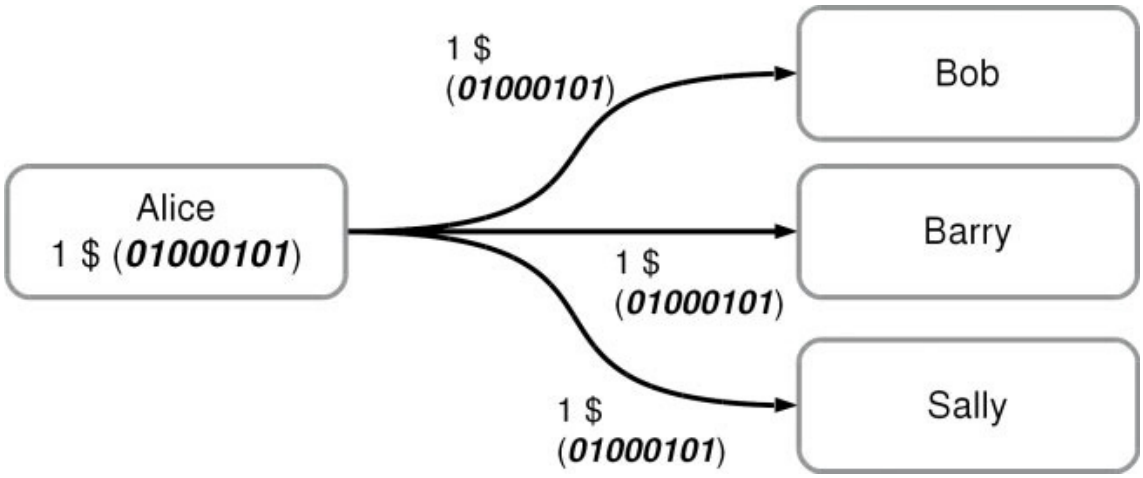

Figura 2.1: Double-spending problem, da "Understanding Bitcoin" [9] figura 2.1

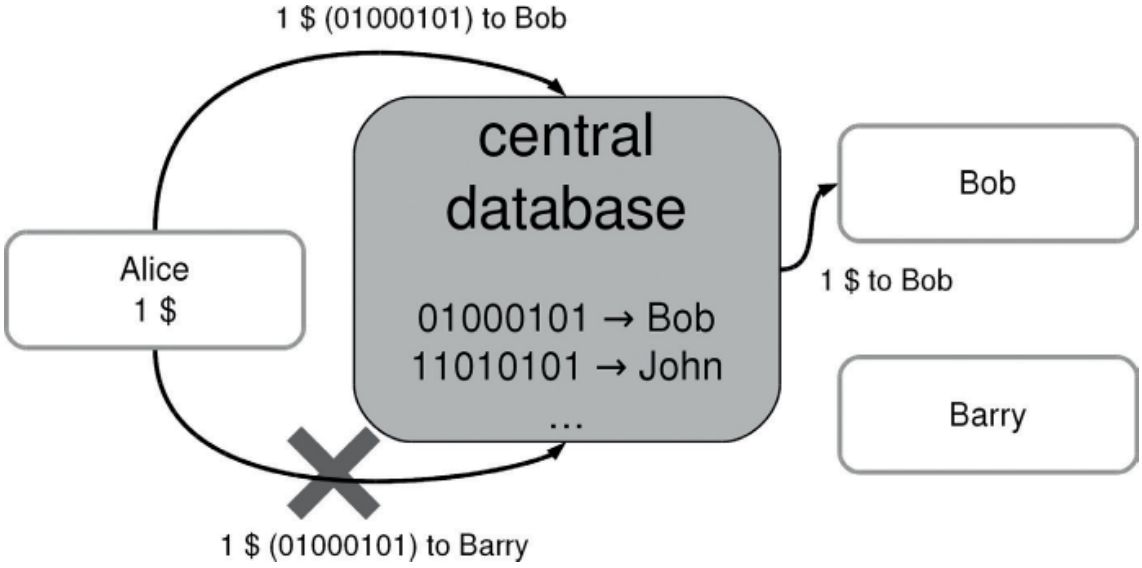

Figura 2.2: Central counterparty holding a centralized database, da "Understanding Bitcoin" [9] figura 2.2

2.1.1 Database centralizzato

La soluzione contro il *double-spending* finora adottata è stata l'affidamento delle transazioni economiche digitali ad un ente esterno fidato (e.g. i sistemi e-banking) che conoscesse le identità degli utenti e potesse verificarne l'effettiva disponibilità economica. È il database centrale che traccia la storia delle transazioni effettuate da ogni utente e prima di autorizzare la successiva verifica che non ci siano incongruenze. Affidarsi ad un sistema centralizzato risolve il problema, ma porta con sé una serie di criticità.
Prima di tutto la necessità di *trust*, fiducia, nei confronti dell'intermediario centrale: esso infatti ha libero accesso ai dati degli utenti e alla loro storia transazionale, si occupa dell'affidabilità dell'intero sistema, può autorizzare o negare transazioni e bloccare interi utenti. Si tratta di un nodo cruciale per la sicurezza del sistema, un attacco riuscito nei suoi confronti comporterebbe conseguenze disastrose. Inoltre, il database rappresenta un *single point of failure*: abbattere il nodo centrale significa abbattere l'intera rete.

2.1.2 Sistemi distribuiti

Un'alternativa ad un sistema controllato da un intermediario centrale è rappresentata da un sistema distribuito decentralizzato. In un sistema distribuito due o più nodi collaborano per svolgere un compito prefissato in maniera trasparente all'utente, che si interfaccia con un'unica piattaforma logica. Emerge la necessità di coordinare i diversi attori: i nodi possono essere onesti e funzionare in modo corretto oppure difettosi, mal funzionanti o maliziosi. Anche se qualche nodo si rivelasse difettoso o la connessione venisse a mancare, un buon sistema distribuito dovrebbe assorbire l'imprevisto e continuare a lavorare senza problemi. La complessità di progettazione di sistemi di questo tipo non è indifferente, è stata area di ricerca fertile per molti anni e lo è tuttora. Un risultato importante è il teorema CAP, che dimostra come non possano coesistere tutte le caratteristiche desiderate in uno stesso sistema.

Teorema CAP

Altrimenti noto come Teorema di Brewer, proposto da Eric Brewer nel 1998 e dimostrato poi da Seth Gilbert e Nancy Lynch nel 2002 [10], il teorema CAP afferma che un qualsiasi sistema distribuito non può garantire simultaneamente le tre seguenti proprietà:

- **Coerenza (Consistency)** è la proprietà che assicura che tutti i nodi del sistema posseggono la stessa copia aggiornata dei dati;
- **Disponibilità (Availability)** indica che il sistema è attivo, accessibile e pronto a ricevere input e a fornire output corretti nei tempi previsti;
- **Tolleranza di partizione (Partition tolerance)** assicura che anche nel caso un gruppo di nodi crollasse per qualche motivo, il sistema continuerebbe ad operare correttamente.

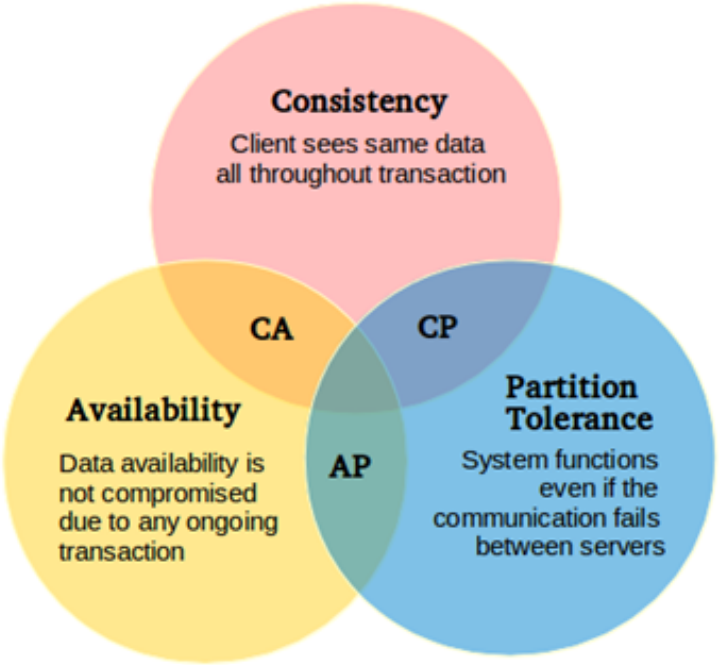

Figura 2.3: Teorema CAP [8]

Problema dei Generali Bizantini

Si tratta di un quesito posto da Leslie Lamport (con Marshall Pease e Robert Shostak) nel 1982 [13], dove un gruppo di generali a capo di diverse sezioni dell'esercito bizantino sta pianificando di attaccare una città. L'unico modo a loro disposizione per comunicare è mediante messaggeri, e hanno bisogno di concordare il momento dell'attacco per vincere. Potrebbero esserci traditori tra di loro, i quali comunicherebbero messaggi discordanti per minare la riuscita dell'operazione: è quindi necessario stabilire un sistema affidabile per raggiungere il consenso sulle tempistiche di attacco. È facile l'analogia con un sistema informatico distribuito in cui i

generali siano rappresentati dai nodi e i messaggeri come un'infrastruttura di rete. Una soluzione praticabile del problema dei generali bizantini in un sistema sincrono è stata proposta da Miguel Castro e Barbara Liskov in *Practical Byzantine Fault Tolerance* [7], mentre Bitcoin è la prima implementazione pratica con la sua soluzione basata sulla Proof-of-Work.

Algoritmi di consenso

La ricerca sui modi per raggiungere il consenso tra nodi di un sistema distribuito si è sviluppata molto nel periodo successivo all'introduzione di Bitcoin, pertanto sarebbe quantomeno ottimistico fornire un elenco di metodi che volesse essere esaustivo. È possibile tuttavia individuare due categorie principali di algoritmi:

- **Basati sulla fault-tolerance Bizantina**, si affidano ad un insieme di nodi che si scambiano messaggi firmati. Non richiedono impiego di risorse intensivo, sono affidabili fintanto che $N > 3F$ dove N è il numero di nodi e F è il numero di nodi malevoli;
- **Basati sul riconoscimento di un leader**, mettono i nodi in competizione per essere riconosciuti come leader e il nodo vincitore propone un accordo. Tipicamente richiedono un impiego di risorse considerevole come "gara" e la sicurezza delle loro applicazioni spesso si basa proprio sulla non convenienza di un tale investimento per un attaccante.

2.2 Bitcoin

Capitolo 3

Potenzialità e limiti

3.1 Generalizzazione

In seguito all'esordio di Blockchain assieme a Bitcoin sono state presentate numerose proposte per sviluppi ed implementazioni della tecnologia. Nel seguito di questa sezione si presenta quello che vuole essere uno schema delle caratteristiche salienti di un sistema basato su Blockchain: sono riportati le principali scelte di progettazione che lo caratterizzano e alcune delle soluzioni ad oggi adottate. La ricerca è estremamente attiva a riguardo, perciò questo elenco non sarà e non vuole essere esaustivo: lo scopo è quello di chiarire a che tipologie di sistema si fa riferimento parlando genericamente di *Blockchain*.

3.1.1 Definizione tecnica

Una buona definizione generica di Blockchain è quella proposta da Imran Bashir in *Mastering Blockchain* [5]: essenzialmente si tratta di un registro distribuito peer-to-peer, crittograficamente sicuro, append-only, immutabile (o meglio, estremamente difficile da modificare) e aggiornabile solo previo consenso da parte dei peer. Esistono ovviamente molte altre definizioni, ciascuna con enfasi diversa sui vari aspetti di Blockchain, tuttavia l'estrema sintesi di quella di Bashir permette di concentrarsi sugli aspetti imprescindibili che la caratterizzano, lasciando alle scelte di progettazione successive il compito di determinarne i particolari.

3.1.2 Accesso

La prima grande suddivisione in macro categorie dei sistemi basati su Blockchain ne riguarda le modalità di accesso. La Blockchain può essere:

- **Pubblica** nel momento in cui il sistema è aperto al pubblico e chiunque può installarne un client, scaricare il registro e partecipare attivamente ai processi di validazione dei blocchi. Tipicamente ogni utente scarica e mantiene una copia locale del registro e tramite un sistema di consenso distribuito si raggiunge un accordo con il resto della rete sull'effettivo stato "ufficiale" dei dati. Tipicamente si affidano ad algoritmi di consenso basati su proof-of-work o proof-of-stake. Sono anche dette *permissionless ledgers*, e l'esempio più noto è Bitcoin;

- **Privata** nel caso l'accesso sia ristretto ad un limitato insieme di utenti o organizzazioni che abbiano deciso di condividere dei registri comuni. In questa accezione di Blockchain acquista importanza l'autenticazione degli utenti, e si ottiene una maggiore riservatezza dei dati a costo di rinunciare alla natura completamente "*trustless*" delle blockchain pubbliche. Ciò porta a differenze sostanziali nella gestione del consenso: è possibile prescindere da prove onerose come la proof-of-work, e soprattutto viene a mancare la necessità di rendere vantaggioso a ciascun peer il mantenimento del sistema. Il grado di chiusura di questo tipo di blockchain può variare anche molto: è possibile ad esempio permettere a ciascun utente di osservare il registro ma limitarne ad un insieme ristretto la modifica. Sono anche dette *permissioned ledgers* e esempi possibili sono Ripple e i sistemi costruiti su Hyperledger.

3.1.3 Consenso

La tipologa di consenso adottata caratterizza fortemente l'intero sistema Blockchain, ed è strettamente legata alla tipologia di accesso scelta.

- **Proof of work**: è il meccanismo di consenso adottato da Bitcoin. Richiede la soluzione ad un problema matematico computazionalmente impegnativo (e.g. *Hashcash*) come prova delle risorse investite per far accettare la propria proposta dalla rete;

- **Proof of stake**: è il meccanismo di consenso usato da Peercoin (e si parla di adottarlo prossimamente in Ethereum [6]). Sbilancia la probabilità di far accettare una proposta di cambiamento del registro in favore dei nodi di maggiore "importanza" all'interno della rete. In una Blockchain che rappresenti una valuta, ad esempio, l'importanza consiste nella quantità di valuta posseduta. Nella variante dei consensi *deposit-based* invece si acquista importanza versando quella che è in pratica una caparra. L'idea di base è che un attaccante dovrebbe investire talmente tante risorse nell'oggetto del suo attacco da renderlo non profittevole, in quanto le ricadute sul suo investimento sarebbero troppo pesanti. Una variante è la *Delegated Proof of Stake* in cui i nodi importanti delegano la validazione di ciascuna transazione tramite votazione;

- **Proof of elapsed time**: è un meccanismo di consenso introdotto da Intel [11] allo scopo di limitare l'enorme dispendio di energie dei sistemi proof-of-work. Utilizza hardware proprietario certificato per garantire l'affidabilità di un timer, ottenendo di fatto lo stesso effetto dei PoW con irrisorio consumo di energia;

- **Reputation based**: è una variante dei sistemi proof-of-stake dal nome autoesplicativo. Il leader viene eletto sulla base della reputazione che si è costruito nel tempo all'interno della rete, che può basarsi su lavoro effettuato o su espressione degli altri nodi;

- **Federated (Byzantine) agreement**: è un sistema non-trustless, e pertanto adatto a blockchain di tipo permissioned. I nodi tengono traccia di un gruppo di peer pubblicamente riconosciuti come affidabili e propaga solo le transazioni validate dalla maggioranza dei nodi affidabili. È implementato nello Stellar Consensus Protocol [14];

- **Practical Byzantine Fault Tolerance**: è la soluzione originale al problema dei generali bizantini proposta da Castro e Liskov nel 1999 [7].

3.1.4 Struttura

La scelta è in questo caso tra:

- Tokenized blockchain;
- Tokenless blockchain.

Blockchain è nata come supporto ad un sistema di scambio di token, e le applicazioni al momento più diffuse rispecchiano questo legame nei confronti di un'unità minima di informazione trasferibile. Soprattutto nel caso di blockchain permissioned, però, è possibile prescindere dal concetto di token, condividendo informazioni analogamente a quello che si può fare mediante un database tradizionale. Hyperledger permette di implementare sistemi di questo tipo, basati su un database condiviso (lo *state*) in cui ogni modifica è registrata in una blockchain che funge da "log certificato".

3.1.5 Indipendenza

Un ultima distinzione degna di nota è quella tra blockchain stand-alone e sidechain [4]. Finora, le implementazioni di sidechain riguardano esclusivamente le tokenized blockchain. Una sidechain si appoggia ad una stand-alone e permette i trasferimenti di token dall'una all'altra. Si appoggia su varianti di proof-of-stake in cui è necessario impegnare dei coin di una catena per generarne nell'altra. I trasferimenti possono essere unidirezionali o bidirezionali. L'interesse sulle sidechain è estremamente alto poiché queste abilitano la creazione di monete che si appoggiano a criptovalute diffuse come Bitcoin e ne vanno a colmare delle lacune. Interessante a proposito il punto di vista di Ferdinando Ametrano sull'uso di sidechain per ottenere la stabilità dei prezzi con una criptovaluta, riportata nel suo paper del 2014 "*Hayek Money: the Cryptocurrency Price Stability Solution*" [1].

3.1.6 Teorema CAP e Blockchain

Può sembrare che le varie implementazioni di Blockchain illustrate violino il teorema CAP presentato in 2.1.2. Ciò è inesatto: in particolare Blockchain persegue disponibilità e tolleranza di partizione in maniera istantanea, mentre la coerenza è sacrificata e ottenuta solo attraverso un lasso di tempo. Si parla in questo caso di *coerenza eventuale*, ed è il motivo per cui il protocollo Bitcoin è stato di fatto artificialmente rallentato mediante un problema a difficoltà dinamica, che assicurasse un impegno per la proof-of-work di circa 10 minuti. Un qualsiasi sistema Blockchain può dirsi coerente solo facendo riferimento a transazioni con una certa "età", che siano state accettate e validate col tempo dai vari peer.

3.2 Benefici e limitazioni

Spunto di riflessione: blockchain innovazione sia in ambito economico che in ambito informatico: rivoluziona nel primo caso, innova nel secondo.
LINK: Ending the bitcoin vs blockchain debate

Nella gran quantità di materiale prodotto su Blockchain dalla sua presentazione ad oggi si contrappongono le posizioni di chi la considera la base della "quarta rivoluzione industriale" [15], o il quinto "disruptive computing paradigm" [17], a chi ne critica fortemente l'efficacia sostenendo che sia applicabile unicamente ai sistemi di cripto valute [3].
L'analisi della questione è complicata dall'intrinseca interdisciplinarità dell'argomento. Ferdinando Ametrano parla di Bitcoin [2] come argomento all'incrocio di quattro discipline: crittografia, reti di calcolatori e trasmissione dati, teoria dei giochi e teoria economica e monetaria. Anche affrontando solamente l'aspetto tecnologico di Blockchain intesa in senso più generale, è fondamentale rendersi conto della moltitudine di sfaccettature possibili: la maggior parte dei confronti tende a semplificare e ad ignorarne alcune in favore di altre, il che conduce a conclusioni completamente diverse. Un altro punto delicato riguarda le enormi differenze tra i diversi tipi di blockchain: sebbene l'affermarsi di questa tecnologia porti indubbiamente *innovazione*, non sempre si può parlare di *rivoluzione* vera e propria. Per comprenderne benefici e limitazioni in maniera equilibrata, è necessario distinguere almeno le due grandi famiglie di blockchain: *permissionless* e *permissioned*.

3.2.1 Blockchain permissioned

Rappresentano l'innovazione più mite. Il controllo operato sui partecipanti al sistema permette di ridurre molti aspetti negativi della tecnologia, ma ne limita certamente il potenziale. In particolare, si può pensare a questi sistemi come ad alternative ai database tradizionali, con l'aggiunta di alcune desiderabili proprietà. Tutti i punti riportati di seguito sono applicabili anche ai sistemi permissionless e pertanto si possono considerare come caratteristiche "generali" di Blockchain.

Vantaggi: [**bozza**]

- **Semplificazione del paradigma attuale**: al momento è possibile ottenere gran parte dei vantaggi con altri sistemi, usare blockchain serve a semplificare e uniformare in un'unica architettura coerente il tutto;
- **Trasparenza**: ogni peer può controllare ogni transazione all'occorrenza, ideale negli scenari in cui sia necessario verificare il rispetto di norme condivise;
- **Condivisibilità**: la natura distribuita del sistema porta con se come diretta conseguenza la condivisione dei suoi contenuti, pur mantenendone pieno controllo sulle modifiche;
- **Immutabilità**: estrema difficoltà di modificare dati scritti in precendenza (impossibilità di fatto). Nel caso particolare dei sistemi permissioned, bisogna fare attenzione a non minare la sicurezza del sistema generale dando per scontata l'affidabilità degli attori;
- **Sicurezza**: tutte le transazioni in una blockchain sono crittograficamente protette e contribuiscono ad assicurare l'integrità del sistema;
- **Disponibilità del servizio**: basarsi su una rete peer-to-peer permette di garantire la disponibilità del servizio anche nel caso un certo numero di peer non

fosse disponibile. Le policy di consenso influenzano anche questo punto molto pesantemente: se stabilisco che una transazione debba ricevere il consenso da tutti gli altri peer, ne basta uno non funzionante e il sistema diventa non disponibile;

Svantaggi: [bozza]

- **Inefficienza rispetto ai database tradizionali**
- **Scalabilità**
- **Complessità e carenza di know-how**
- **Tecnologia relativamente immatura**

3.2.2 Blockchain pubbliche

Rappresentano l'aspetto rivoluzionario di Blockchain. L'idea originaria di Nakamoto ha permesso per la prima volta di generare un'entità digitale *scarsa*, ovvero non arbitrariamente replicabile, senza la necessità della supervisione di un intermediario centrale. In questa tipologia si trovano i vantaggi più dirompenti, assieme ai limiti più vincolanti.

Vantaggi: [bozza]

- **No single point of failure**
- **Non censurabilità o revocabilità del servizio**
- **Velocità di validazione**: prescindere da intermediari può portare ad un'accorciamento dei tempi necessari allo svolgimento di operazioni come quelle finanziarie;
- **Risparmio sulle transazioni**;

Svantaggi: [bozza]

- **Scalabilità**: situazione estremamente più drastica di quella dei sistemi permissioned;
- **Adattabilità**
- **Regolamentazione**
- **Privacy**
- **Impossibilità di interventi correttivi**

Capitolo 4

Proposta di use case

Alla luce di quanto scritto finora, una possibile applicazione della tecnologia si può individuare nell'e-voting. Nei sistemi di voto attualmente in uso, la correttezza del voto e l'anonimato del votante sono totalmente affidati al controllo di un ente o di un sistema centrale in cui si ha fiducia come può essere lo stato o l'organo che indice la votazione. Questo vincola gli eleggibili e gli elettori ad una mancanza di trasparenza finora ritenuta inevitabile, e il sistema elettorale a dover investire risorse considerevoli nel gestire la votazione a partire da quando viene indetta fino al termine degli scrutini. L'avvento della tecnologia Blockchain permette un cambio di paradigma in favore di sistemi decentralizzati e trasparenti in cui non sia necessario riporre fiducia nell'operato di alcuno degli agenti coinvolti. Si propone quindi un modello che permetta di applicare queste caratteristiche desiderabili ad una votazione elettronica.

4.1 Tecnologie e algoritmi utilizzati

4.1.1 Hyperledger Fabric v.1.0

Hyperledger è un progetto collaborativo della Linux Foundation che mira a creare un framework open source per lo sviluppo di sistemi Blockchain in ambito aziendale. Fabric è uno dei sottoprogetti che fanno parte di Hyperledger, in particolare rappresenta il principale contributo di IBM alla causa. Nella visione di Hyperledger Fabric un sistema Blockchain aziendale deve essere basato su di un'architettura modulare in cui i diversi elementi, come algoritmi di consenso, sistema di verifica delle identità, protocolli di cifratura o tipo di database ausiliari, siano intercambiabili e sostituibili liberamente. Permette anche lo sviluppo di smart contract, qui chiamati chaincode, attraverso l'uso di un qualsiasi linguaggio di programmazione: caratteristica che lo rende estremamente più flessibile del limitato set di istruzioni previsto da Bitcoin o dal linguaggio dedicato sviluppato da Ethereum. Per fare ciò l'esecuzione dei chaincode è isolata in un container dedicato, creato attraverso Docker, contenente un sistema operativo sicuro, il linguaggio del chaincode e gli SDK che permettono di interfacciarsi con il sistema. Fabric è un sistema permissioned, tutti i partecipanti sono tenuti ad autenticarsi attraverso un servizio dedicato (anche questo intercambiabile) per avere accesso alla rete.

Componenti di Fabric

- **Client**: sono gli enti partecipanti alla rete che invocano un chaincode o propongono una transazione. Rappresentano gli utenti finali, non mantengono una copia della blockchain ma devono connettersi con un peer per interagire con il sistema;

- **Peer**: sono gli enti che effettuano le transazioni e mantengono il database condiviso e la blockchain. I peer sono coordinati dall'orderer secondo delle policy specificate a priori;

- **Orderer**: è l'ente (eventualmente distribuito) che sovrintende alla comunicazione tra i nodi della rete. Fornisce garanzie di consegna dei messaggi immessi nella rete, tra cui le proposte di transazione da parte dei peer. È il responsabile del consenso all'interno del sistema distribuito;

- **Membership services**: sono l'astrazione di tutti i meccanismi crittografici e i protocolli che lavorano alla validazione dei certificati e al riconoscimento degli utenti;

- **Organizzazioni**: rappresentano ciascuna entità distinta che possiede un certificato univoco per la rete. I nodi come i peer o i client devono essere legati ad un'organizzazione per poter essere riconosciuti e autenticati;

- **Channel**: è l'overlay della blockchain privata di Hyperledger. Ciascun canale ha un registro specifico, condiviso tra i peer registrati al canale, e ciascun ente coinvolto nella transazione deve essere autenticato ad uno specifico channel affinché questa abbia luogo;

- **State**: abbreviazione per "State Database", è il database in cui è salvato lo stato attuale del sistema per effettuare query sul registro in maniera efficiente. È modellato come un database chiave/valore con versionamento. Può essere completamente ricostruito dalla storia delle transazioni salvata in blockchain, il che ne garantisce il controllo sull'affidabilità;

- **Chaincode**: è il nome dato ai programmi salvati nella blockchain di Hyperledger. Sono invocati dalle transazioni, e possono inizializzare componenti nello state o modificarli. La loro esecuzione avviene in un container Docker isolato per garantirne la riservatezza. Incapsula la componente di business logic del sistema, ha corrispondenza diretta con quelli che in altri sistemi basati su Blockchain vengono chiamati *smart contract*.

- **Transazioni**: sono la rappresentazione di ogni azione effettuata sul registro. Possono essere *deploy transactions* che creano nuovo chaincode e lo "installano" in un canale oppure *invoke transactions* che invocano una funzione presente in un chaincode installato per effettuare letture o modifiche al ledger. Ciascuna transazione deve essere approvata e gestita dall'orderer, e viene registrata nella blockchain per la futura tracciabilità.

4.1.2 Ring signature

Una ring signature, o firma ad anello, è un tipo di firma digitale legata ad un gruppo di utenti ciascuno in possesso di una chiave. Questo tipo di firma permette di verificare che l'autore appartiene a quel dato gruppo di utenti rendendo però computazionalmente infattibile determinare quale specifico utente esso sia. Riporto di seguito (tradotta) una possibile definizione formale [18]:
Si supponga che un gruppo di entità possieda le coppie di chiavi pubbliche/private $(P_1, S_1), (P_2, S_2), ..., (P_n, S_n)$. Il soggetto i può apporre una ring signature σ sul messaggio m avendo in input $(m, S_i, P_1, ..., P_n)$. Chiunque può verificare la validità della firma dati σ, m e le chiavi pubbliche coinvolte $P_1, ..., P_n$.

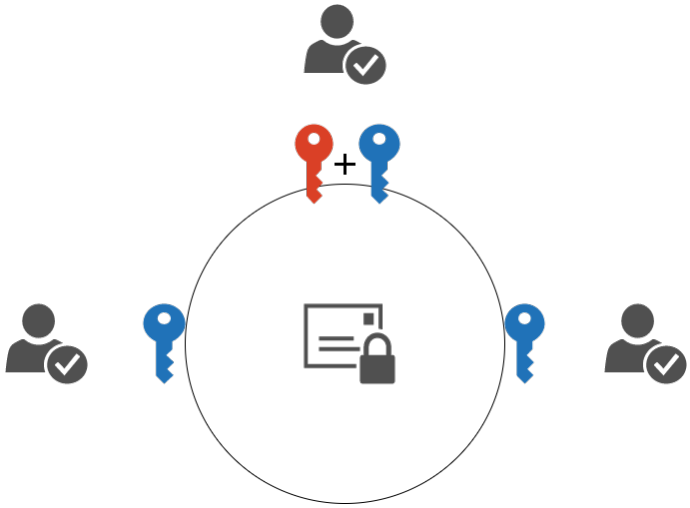

Figura 4.1: Ring signature

4.2 Architettura

Il sistema si articola in due blockchain separate, la blockchain di firma e la blockchain di voto (da ora rispettivamente Signature chain o SC e Vote chain o VC). Le due chain sono isolate, per garantire la non collegabilità del voto al votante in fase di scrutinio. Le due chain hanno scopi diversi: SC assicura che il voto avvenga in maniera controllata e abilita al voto il client, inoltre contiene le informazioni per dividere in raggruppamenti i votanti similmente a quanto avviene con le sezioni elettorali; VC raccoglie i voti e li registra per effettuare poi il conteggio, ma deve garantire l'anonimato del votante.

4.2.1 Preparazione

Viene indetta una votazione. Si prevede che l'architettura sia già predisposta, in particolare devono essere noti ed affidabili, completi di chiavi pubbliche:

- L'elenco dei votanti;
- L'elenco dei peer;
- L'elenco degli indirizzi dei candidati per la Vote chain;

- L'elenco dei programmi client verificati.

Inizializzazione della Signature chain

La SC viene predisposta con il chaincode che permette al client di interrogare la chain per conoscere le chiavi pubbliche dei votanti inseriti nello stesso raggruppamento, registrando contemporaneamente la propria firma. I dati sui raggruppamenti dei votanti sono inseriti nella SC. Questa dovrà essere accessibile solamente ai peer deputati al controllo della votazione, a causa della riservatezza delle informazioni sui raggruppamenti.

Inizializzazione della Vote chain

La VC contiene inizialmente delle transazioni che registrano l'associazione di ogni indirizzo numerico ad un nome intelligibile del candidato corrispondente, i saldi dei voti dei candidati inizializzati a 0 e il chaincode necessario al client per incrementare di 1 il saldo del candidato scelto. Un altro chaincode utile è quello che permette ai peer di ricevere il conteggio finale e quindi l'esito della votazione, così come il chaincode che permette ad un servizio web di interrogare la blockchain e rendere pubblici i dati in essa contenuti per permettere al votante la verifica del voto espresso.

4.2.2 Workflow di una votazione

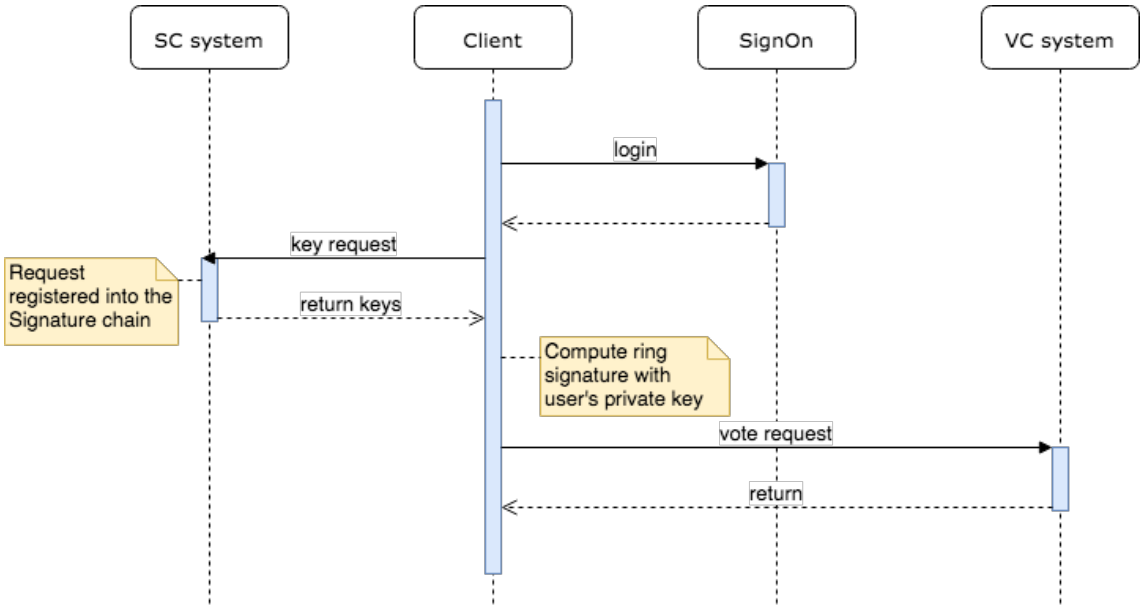

Figura 4.2: Workflow della votazione

Il diagramma in figura 4.2 illustra il workflow tipico di una votazione.

Signature chain: Firma del registro votanti

Il client autentica l'utente attraverso un sign-on affidabile. I controlli di accesso possono essere rinforzati o meno da altri passaggi affidati a sistemi di intelligenza artificiale o al controllo di un operatore. A utente autenticato, il client invoca

il chaincode di query su SC ed ottiene l'elenco delle chiavi pubbliche del suo raggruppamento. Questa richiesta viene registrata in blockchain, per assicurarsi che lo stesso utente non possa ripetere la procedura di voto due volte.

Vote chain: In cabina elettorale

Il client si trova ora in possesso delle chiavi pubbliche della sua sezione. In locale permette al votante di esprimere la propria preferenza di voto e richiede la transazione corrispondente nella VC firmandola con la ring signature del proprio raggruppamento. La votazione è ora conclusa.

Scrutinio e verifica

Le operazioni di scrutinio e verifica possono ora essere completamente automatizzate, facendo riferimento ad una base di dati sicura e virtualmente inalterabile come la blockchain.

4.3 Analisi

In questa sezione si analizzano le scelte architetturali fatte argomentandole, concludendo con delle possibilità di miglioramento che si potranno presentare con l'avanzamento dello scenario tecnologico attuale.

4.3.1 Qualità del voto

Il voto è definito dall'art. 48 [12] della Costituzione Italiana come "personale ed eguale, libero e segreto".

Personalità

La personalità del voto, intesa come la necessità di esercitarlo di persona e non tramite terzi, è delegata in questa proposta al servizio di sign-on. Al momento attuale, è possibile fare affidamento su diversi sistemi di autenticazione: per il corretto funzionamento del sistema è previsto che ogni utente abbia una coppia di chiavi pubblica e privata distribuita precedentemente, sotto verifica di un operatore. Ad esempio, è possibile distribuire queste chiavi contestualmente alla consegna della tessera elettorale. La natura certificata di queste chiavi le rende ideali come metodo di autenticazione ma è possibile aggiungere ulteriori controlli come una verifica tramite dati biometrici.

Eguaglianza

L'eguaglianza del voto consiste nell'avere ogni voto lo stesso valore di tutti gli altri. Si articola nell'impedire, in particolare, il voto plurimo e il voto multiplo. Nel sistema proposto, il voto plurimo è scongiurato dal codice del chaincode della Vote chain, il quale permette solo incrementi unitari al saldo dei candidati. Uno scenario di voto multiplo è invece controllato dalla Signature chain e dalla struttura del client: è infatti indispensabile che la procedura di voto sia strettamente successiva alla procedura di firma in SC, e non invocabile in maniera indipendente. In tal modo è

possibile garantire che una stessa persona non reiteri l'esecuzione del chaincode di voto. Pur ammettendo il caso in cui un attaccante riesca a violare il client, questo potrà reiterare voti firmati dalla stessa ring signature: un controllo automatico può facilmente accorgersi della non coerenza del numero di votanti nel raggruppamento e il numero di voti espressi dal raggruppamento, invalidando di fatto le transazioni ma mantenendo confinato l'evento disastroso: si possono applicare le procedure invocate attualmente qualora venissero riscontrate irregolarità che compromettano la validità dei voti di una sezione elettorale.

Libertà

La libertà del voto viene garantita dal negare la possibilità di voto qualora il votante fosse stato costretto con mezzi illeciti a esprimere una certa preferenza. Nel sistema di voto attuale è garantita dal controllo degli operatori di seggio. Nei sistemi di voto remoto l'argomento risulta spinoso: è infatti estremamente difficile stabilire una situazione di particolare pressione dell'elettore, o l'interferenza di esterni. Nel caso di voto elettronico, forzare il client ufficiale a poter essere installato solo su dispositivi muniti di telecamera frontale potrebbe risolvere il problema. Il controllo può essere svolto da appositi addetti eventualmente aiutati da un'intelligenza artificiale simile a quella utilizzata da Unilever [16] per i colloqui di lavoro. Questo non scongiura ogni possibile manipolazione, ma va evidenziato che il controllo sul sistema proposto non lo rende meno sicuro del sistema già in essere: permette anzi un livello di garanzia estremamente più alto di quello dei sistemi di voto per corrispondenza, già accettati in Italia in caso di votanti dall'estero e adottato in maniera diffusa in Svizzera.

Segretezza

La segretezza del voto nel sistema proposto ha come fulcro la completa separazione delle due blockchain. Affidando al client il compito di mantenere la continuità dell'operazione di voto, l'accoppiamento tra votante e voto espresso non può essere ricostruito con certezza in alcun modo. L'unica informazione che può trapelare riguarda il momento in cui viene registrata la votazione nelle due chain, tuttavia questa informazione è molto meno affidabile di quanto possa sembrare a causa della consistenza eventuale della blockchain. Sebbene qualche informazione trapeli, l'elevato numero di transazioni contemporanee previsto e l'incertezza nel determinare l'esatto momento di voto rende di fatto inservibili queste informazioni.

4.3.2 Necessità dei raggruppamenti

La suddivisione in sezioni elettorali è assolutamente necessaria nel sistema di voto attualmente adottato a causa di problemi logistici: è impensabile infatti raggruppare tutte le schede e svolgere un'unica operazione di scrutinio, soprattutto considerando il livello di sorveglianza che sarebbe richiesto in ogni momento dell'operazione. In un sistema automatico come quello proposto, esente da queste problematiche, sono stati introdotti raggruppamenti di elettori per differenti ragioni. In primo luogo, la complessità computazionale degli algoritmi più noti e affermati per la ring signature è lineare: questo preclude l'uso significativo di questa tecnologia (centrale nel modello proposto) per insiemi di chiavi sufficientemente grandi. In secondo luogo, nel

caso un attacco permettesse di replicare lo stesso voto più volte la divisione in raggruppamenti permette di contenere i danni al solo raggruppamento interessato. È da sottolineare come, nel caso di un sistema elettronico, l'indipendenza dai problemi logistici dei sistemi tradizionali permetta di variare di volta in volta la composizione dei raggruppamenti, rendendo inutile qualsiasi informazione trapelata dalle votazioni precedenti. Questa è una condizione certamente più sicura di quella attuale, dove una semplice osservazione sul posto permette di identificare gli appartenenti a ciascuna sezione e far trapelare quindi informazioni (per quanto parziali) sulle preferenze di voto espresse: essendo i risultati delle votazioni per seggio pubblici, il trapelare di qualsiasi informazione sulla composizione del seggio è da considerarsi quantomeno indesiderato.

4.3.3 Prevedibili miglioramenti futuri al sistema

Al momento passaggi critici come l'autenticazione utente e il passaggio dalla SC alla VC devono essere svolti off-chain, ma future scoperte potrebbero aprire la strada verso implementazioni più complete. Portare l'autenticazione utente su Blockchain è un progetto che si lega con la creazione di un sistema di identità digitale decentralizzata. Immaginando un futuro (ad ora remoto) in cui siano disponibili documenti affidabili on-chain attraverso qualche tecnologia, l'autenticazione utente potrebbe svolgersi in questo modo guadagnando le caratteristiche di decentralità e sicurezza proprie della Blockchain.
Più vicino alla situazione attuale è un meccanismo che permetta di abilitare alla votazione su VC direttamente da SC, senza rinunciare all'anonimato del votante. Al momento, ogni chaincode di Hyperledger è eseguito in un ambiente completamente isolato per ragioni di sicurezza. Sono già previste successive implementazioni che permettano a diversi chaincode di interagire tra di loro. Questo aprirebbe ad una modifica del sistema di voto proposto: sarebbe possibile implementare su VC un sistema di token: questi token sarebbero generati quando viene indetta la votazione in base al numero di aventi diritto al voto, e versati dai chaincode verso dei "portafogli" temporanei da loro generati casualmente. Questi sarebbero poi passati al client già cifrati in maniera da mantenere chiunque, eccetto il software del chaincode, all'oscuro dell'accoppiamento votante-portafoglio. In questo modo sarebbe possibile prescindere dai raggruppamenti affidando la sicurezza del sistema unicamente alla solidità della Blockchain. L'aspetto negativo è che si perderebbe la capacità dei raggruppamenti di confinare un eventuale attaccante.

Capitolo 5

Conclusioni

Appendice A

Scenario attuale

https://appliedblockchain.com/outstanding-challenges-in-blockchain-2017/

Appendice B

Algoritmi promettenti

B.1 Algoritmi zero-knowledge

B.1.1 Zero-Knowledge Password Proof

`https://en.wikipedia.org/wiki/Zero-knowledge_password_proof` in una futuristica implementazione potrebbero permettere di dare una key di voto all'utente, che potrebbe usarla per votare senza renderla palese alla VC.

B.1.2 Piattaforma Enigma

`https://www.enigma.co/enigma_full.pdf` permetterebbe modifiche del sistema in grado di garantire riservatezza totale del voto fino al momento della votazione, o scenari prima impensabili come l'identificazione del vincitore senza dare informazioni sul conteggio finale dei voti.

B.1.3 zk-SNARK

`https://z.cash/technology/zksnarks.html`

Appendice C

Implementazione

C.1 Costruzione della rete

```
#################################################################################
#
#   Lancia la rete
#   Devono essere pronti e preconfigurati i seguenti file:
#       - crypto-config.yaml
#       - configtx.yaml
#       - docker-compose.yml
#           - al momento e' commentato, vuole del chaincode in ./chaincode
#
#################################################################################

mkdir crypto-config
cryptogen generate --config=./crypto-config.yaml
# crea rispettive chiavi nella cartella ./crypto-config

export FABRIC_CFG_PATH=$PWD
mkdir channel-artifacts
configtxgen -profile EurasiaOrdererGenesis -outputBlock ./channel-artifacts/genesis
    .block
configtxgen -profile SignatureChannel -outputCreateChannelTx ./channel-artifacts/
    signature_channel.tx -channelID sigChannel
configtxgen -profile VoteChannel -outputCreateChannelTx ./channel-artifacts/
    vote_channel.tx -channelID vtChannel

# Fisso gli anchor peer per ogni organizzazione in ciascun canale
configtxgen -profile SignatureChannel -outputAnchorPeersUpdate ./channel-artifacts/
    sigStatoMSPanchors.tx -channelID sigChannel -asOrg Stato
configtxgen -profile VoteChannel -outputAnchorPeersUpdate ./channel-artifacts/
    vtStatoMSPanchors.tx -channelID vtChannel -asOrg Stato
configtxgen -profile VoteChannel -outputAnchorPeersUpdate ./channel-artifacts/
    vtRepubblicaniMSPanchors.tx -channelID vtChannel -asOrg Repubblicani
configtxgen -profile VoteChannel -outputAnchorPeersUpdate ./channel-artifacts/
    vtMonarchiciMSPanchors.tx -channelID vtChannel -asOrg Monarchici

# Butto su tutto
# Exit on first error, print all commands.
set -ev
# don't rewrite paths for Windows Git Bash users
export MSYS_NO_PATHCONV=1
docker-compose -f docker-compose.yml down
TIMEOUT=72000 docker-compose -f docker-compose.yml up -d

# wait for Hyperledger Fabric to start
# incase of errors when running later commands, issue export FABRIC_START_TIMEOUT=<
    larger number>
export FABRIC_START_TIMEOUT=10
#echo ${FABRIC_START_TIMEOUT}
sleep ${FABRIC_START_TIMEOUT}
```

```
#================================================================================
# CREO I CHANNEL E FACCIO IL JOIN
#================================================================================
#--------------------------------------------------------------------------------
# per peer0.Stato
#--------------------------------------------------------------------------------
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/
    crypto/peerOrganizations/Stato.eurasia.com/users/Admin@Stato.eurasia.com/msp
export CORE_PEER_ADDRESS=peer0.Stato.eurasia.com:7051
export CORE_PEER_LOCALMSPID="StatoMSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/peerOrganizations/Stato.eurasia.com/peers/peer0.Stato.eurasia.com/
    tls/ca.crt
#
#   sigChannel
#
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel create -o orderer.eurasia.com
    :7050 -c sigChannel -f ./channel-artifacts/signature_channel.tx --tls
    $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/ordererOrganizations/eurasia.com/orderers/orderer.eurasia.com/msp/
    tlscacerts/tlsca.eurasia.com-cert.pem
#   anchor peer
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel update -o orderer.eurasia.com
    :7050 -c sigChannel -f ./channel-artifacts/sigStatoMSPanchors.tx --tls
    $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/ordererOrganizations/eurasia.com/orderers/orderer.eurasia.com/msp/
    tlscacerts/tlsca.eurasia.com-cert.pem
#   join
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel join -b sigChannel.block
#
#   vtChannel
#
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel create -o orderer.eurasia.com
    :7050 -c vtChannel -f ./channel-artifacts/vote_channel.tx --tls
    $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/ordererOrganizations/eurasia.com/orderers/orderer.eurasia.com/msp/
    tlscacerts/tlsca.eurasia.com-cert.pem
#   anchor peer
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel update -o orderer.eurasia.com
    :7050 -c vtChannel -f ./channel-artifacts/vtStatoMSPanchors.tx --tls
    $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/ordererOrganizations/eurasia.com/orderers/orderer.eurasia.com/msp/
    tlscacerts/tlsca.eurasia.com-cert.pem
#   join
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel join -b vtChannel.block
#
#
#--------------------------------------------------------------------------------
# per peer0.Repubblicani
#--------------------------------------------------------------------------------
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/
    crypto/peerOrganizations/Repubblicani.eurasia.com/users/Admin@Repubblicani.
    eurasia.com/msp
export CORE_PEER_ADDRESS=peer0.Repubblicani.eurasia.com:7051
export CORE_PEER_LOCALMSPID="RepubblicaniMSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/peerOrganizations/Repubblicani.eurasia.com/peers/peer0.Repubblicani
    .eurasia.com/tls/ca.crt
#
#   vtChannel
#
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel create -o orderer.eurasia.com
    :7050 -c vtChannel -f ./channel-artifacts/vote_channel.tx --tls
    $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/ordererOrganizations/eurasia.com/orderers/orderer.eurasia.com/msp/
    tlscacerts/tlsca.eurasia.com-cert.pem
```

```
#   anchor peer
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel update -o orderer.eurasia.com
    :7050 -c vtChannel -f ./channel-artifacts/vtRepubblicaniMSPanchors.tx --tls
    $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/ordererOrganizations/eurasia.com/orderers/orderer.eurasia.com/msp/
    tlscacerts/tlsca.eurasia.com-cert.pem
#   join
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel join -b vtChannel.block
#
#
#-------------------------------------------------------------------------------
# per peer0.Monarchici
#-------------------------------------------------------------------------------
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/
    crypto/peerOrganizations/Monarchici.eurasia.com/users/Admin@Monarchici.eurasia.
    com/msp
export CORE_PEER_ADDRESS=peer0.Monarchici.eurasia.com:7051
export CORE_PEER_LOCALMSPID="MonarchiciMSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/peerOrganizations/Monarchici.eurasia.com/peers/peer0.Monarchici.
    eurasia.com/tls/ca.crt
#
#   vtChannel
#
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel create -o orderer.eurasia.com
    :7050 -c vtChannel -f ./channel-artifacts/vote_channel.tx --tls
    $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/ordererOrganizations/eurasia.com/orderers/orderer.eurasia.com/msp/
    tlscacerts/tlsca.eurasia.com-cert.pem
#   anchor peer
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel update -o orderer.eurasia.com
    :7050 -c vtChannel -f ./channel-artifacts/vtMonarchiciMSPanchors.tx --tls
    $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/
    peer/crypto/ordererOrganizations/eurasia.com/orderers/orderer.eurasia.com/msp/
    tlscacerts/tlsca.eurasia.com-cert.pem
#   join
docker exec -e CORE_PEER_LOCALMSPID -e CORE_PEER_MSPCONFIGPATH -e CORE_PEER_ADDRESS
     -e CORE_PEER_TLS_ROOTCERT_FILE cli peer channel join -b vtChannel.block
```

C.2 Configurazione crypto-tool

```
#-------------------------------------------------------------------------------
#   Contenuto del file crypto-config.yaml
# ---------------------------------------------------------------------------
# ---------------------------------------------------------------------------
# "OrdererOrgs" - Definition of organizations managing orderer nodes
# ---------------------------------------------------------------------------
OrdererOrgs:
  # ---------------------------------------------------------------------------
  # Orderer
  # ---------------------------------------------------------------------------
  - Name: Orderer
    Domain: eurasia.com
    # ---------------------------------------------------------------------------
    # "Specs" - See PeerOrgs below for complete description
    # ---------------------------------------------------------------------------
    Specs:
      - Hostname: orderer
# ---------------------------------------------------------------------------
# "PeerOrgs" - Definition of organizations managing peer nodes
# ---------------------------------------------------------------------------
PeerOrgs:
  # ---------------------------------------------------------------------------
  # Org1
  # ---------------------------------------------------------------------------
```

```
  - Name: Stato
    Domain: Stato.eurasia.com
    # ---------------------------------------------------------------------------
    # "Specs"
    # ---------------------------------------------------------------------------
    # Uncomment this section to enable the explicit definition of hosts in your
    # configuration.  Most users will want to use Template, below
    #
    # Specs is an array of Spec entries.  Each Spec entry consists of two fields:
    #   - Hostname:   (Required) The desired hostname, sans the domain.
    #   - CommonName: (Optional) Specifies the template or explicit override for
    #                 the CN.  By default, this is the template:
    #
    #                            "{{.Hostname}}.{{.Domain}}"
    #
    #                 which obtains its values from the Spec.Hostname and
    #                 Org.Domain, respectively.
    # ---------------------------------------------------------------------------
    # Specs:
    #   - Hostname: foo # implicitly "foo.org1.example.com"
    #     CommonName: foo27.org5.example.com # overrides Hostname-based FQDN set
      above
    #   - Hostname: bar
    #   - Hostname: baz
    # ---------------------------------------------------------------------------
    # "Template"
    # ---------------------------------------------------------------------------
    # Allows for the definition of 1 or more hosts that are created sequentially
    # from a template. By default, this looks like "peer%d" from 0 to Count-1.
    # You may override the number of nodes (Count), the starting index (Start)
    # or the template used to construct the name (Hostname).
    #
    # Note: Template and Specs are not mutually exclusive.  You may define both
    # sections and the aggregate nodes will be created for you.  Take care with
    # name collisions
    # ---------------------------------------------------------------------------
    Template:
      Count: 1
      # Start: 5
      # Hostname: {{.Prefix}}{{.Index}} # default
    # ---------------------------------------------------------------------------
    # "Users"
    # ---------------------------------------------------------------------------
    # Count: The number of user accounts _in addition_ to Admin
    # ---------------------------------------------------------------------------
    Users:
      Count: 1
  # ---------------------------------------------------------------------------
  # Second organization
  # ---------------------------------------------------------------------------
  - Name: Repubblicani
    Domain: Repubblicani.eurasia.com
    Template:
      Count: 1
    Users:
      Count: 0
  # ---------------------------------------------------------------------------
  # Third organization
  # ---------------------------------------------------------------------------
  - Name: Monarchici
    Domain: Monarchici.eurasia.com
    Template:
      Count: 1
    Users:
      Count: 0
```

C.3 Configurazione configtxgen

```
#---------------------------------------------------------------------------
```

```
#   Contenuto del file configtx.yaml
#-------------------------------------------------------------------------------
---
################################################################################
#
#   Profile
#
#   - Different configuration profiles may be encoded here to be specified
#   as parameters to the configtxgen tool
#
################################################################################
Profiles:
    EurasiaOrdererGenesis:
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
        Consortiums:
            VotingConsortium:
                Organizations:
                    - *Stato
                    - *Repubblicani
                    - *Monarchici
    SignatureChannel:
        Consortium: VotingConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Stato
    VoteChannel:
        Consortium: VotingConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Stato
                - *Repubblicani
                - *Monarchici

################################################################################
#
#   Section: Organizations
#
#   - This section defines the different organizational identities which will
#   be referenced later in the configuration.
#
################################################################################
Organizations:
    - &OrdererOrg
        Name: OrdererOrg
        # ID to load the MSP definition as
        ID: OrdererMSP
        # MSPDir is the filesystem path which contains the MSP configuration
        MSPDir: crypto-config/ordererOrganizations/eurasia.com/msp

    - &Stato
        Name: Stato
        # ID to load the MSP definition as
        ID: StatoMSP
        # MSPDir is the filesystem path which contains the MSP configuration
        MSPDir: crypto-config/peerOrganizations/Stato.eurasia.com/msp
        AnchorPeers:
            # AnchorPeers defines the location of peers which can be used
            # for cross org gossip communication.  Note, this value is only
            # encoded in the genesis block in the Application section context
            - Host: peer0.Stato.eurasia.com
              Port: 7051

    - &Repubblicani
        Name: Repubblicani
        ID: RepubblicaniMSP
        MSPDir: crypto-config/peerOrganizations/Repubblicani.eurasia.com/msp
        AnchorPeers:
            - Host: peer0.Repubblicani.eurasia.com
```

```
            Port: 7051

    - &Monarchici
        Name: Monarchici
        ID: MonarchiciMSP
        MSPDir: crypto-config/peerOrganizations/Monarchici.eurasia.com/msp
        AnchorPeers:
            - Host: peer0.Monarchici.eurasia.com
              Port: 7051

################################################################################
#
#   SECTION: Orderer
#
#   - This section defines the values to encode into a config transaction or
#   genesis block for orderer related parameters
#
################################################################################
Orderer: &OrdererDefaults

    # Orderer Type: The orderer implementation to start
    # Available types are "solo" and "kafka"
    OrdererType: solo

    Addresses:
        - orderer.eurasia.com:7050

    # Batch Timeout: The amount of time to wait before creating a batch
    BatchTimeout: 2s

    # Batch Size: Controls the number of messages batched into a block
    BatchSize:

        # Max Message Count: The maximum number of messages to permit in a batch
        MaxMessageCount: 10

        # Absolute Max Bytes: The absolute maximum number of bytes allowed for
        # the serialized messages in a batch.
        AbsoluteMaxBytes: 99 MB

        # Preferred Max Bytes: The preferred maximum number of bytes allowed for
        # the serialized messages in a batch. A message larger than the preferred
        # max bytes will result in a batch larger than preferred max bytes.
        PreferredMaxBytes: 512 KB

    Kafka:
        # Brokers: A list of Kafka brokers to which the orderer connects
        # NOTE: Use IP:port notation
        Brokers:
            - 127.0.0.1:9092

    # Organizations is the list of orgs which are defined as participants on
    # the orderer side of the network
    Organizations:

################################################################################
#
#   SECTION: Application
#
#   - This section defines the values to encode into a config transaction or
#   genesis block for application related parameters
#
################################################################################
Application: &ApplicationDefaults

    # Organizations is the list of orgs which are defined as participants on
    # the application side of the network
    Organizations:
```

C.4 Compose

```
#
# Contenuto del file docker-compose.yml
#
version: '2'

networks:
  voting:

services:
  ca.eurasia.com:
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca.eurasia.com
    ports:
      - "7054:7054"
    command: sh -c 'fabric-ca-server start --ca.certfile /etc/hyperledger/fabric-ca
        -server-config/Stato.eurasia.com-cert.pem --ca.keyfile /etc/hyperledger/
        fabric-ca-server-config/
        a22daf356b2aab5792ea53e35f66fccef1d7f1aa2b3a2b92dbfbf96a448ea26a_sk -b
        admin:adminpw -d'
    volumes:
      - ./crypto-config/peerOrganizations/Stato.eurasia.com/ca/:/etc/hyperledger/
          fabric-ca-server-config
    container_name: ca.eurasia.com
    networks:
      - voting

  orderer.eurasia.com:
    container_name: orderer.eurasia.com
    image: hyperledger/fabric-orderer
    environment:
      - ORDERER_GENERAL_LOGLEVEL=debug
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_GENESISMETHOD=file
      - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
      # enabled TLS
      - ORDERER_GENERAL_TLS_ENABLED=true
      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
    command: orderer
    volumes:
      - ../channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis
          .block
      - ../crypto-config/ordererOrganizations/eurasia.com/orderers/orderer.eurasia.
          com/msp:/var/hyperledger/orderer/msp
      - ../crypto-config/ordererOrganizations/eurasia.com/orderers/orderer.eurasia.
          com/tls/:/var/hyperledger/orderer/tls
    ports:
      - 7050:7050
    networks:
      - voting

  peer0.Stato.eurasia.com:
    container_name: peer0.Stato.eurasia.com
    image: hyperledger/fabric-peer
    environment:
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      # the following setting starts chaincode containers on the same
      # bridge network as the peers
      # https://docs.docker.com/compose/networking/
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_voting
      #- CORE_LOGGING_LEVEL=ERROR
      - CORE_LOGGING_LEVEL=DEBUG
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_GOSSIP_USELEADERELECTION=true
```

```
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_PROFILE_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
    - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
    - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
    - CORE_PEER_ID=peer0.Stato.eurasia.com
    - CORE_PEER_ADDRESS=peer0.Stato.eurasia.com:7051
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.Stato.eurasia.com:7051
    - CORE_PEER_LOCALMSPID=StatoMSP
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
  command: peer node start
  # command: peer node start --peer-chaincodedev=true
  ports:
    - 7051:7051
    - 7053:7053
  volumes:
    - /var/run/:/host/var/run/
    - ../crypto-config/peerOrganizations/Stato.eurasia.com/peers/peer0.Stato.
        eurasia.com/msp:/etc/hyperledger/fabric/msp
    - ../crypto-config/peerOrganizations/Stato.eurasia.com/peers/peer0.Stato.
        eurasia.com/tls:/etc/hyperledger/fabric/tls
  depends_on:
    - orderer.eurasia.com
  networks:
    - voting

peer0.Repubblicani.eurasia.com:
  container_name: peer0.Repubblicani.eurasia.com
  image: hyperledger/fabric-peer
  environment:
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    # the following setting starts chaincode containers on the same
    # bridge network as the peers
    # https://docs.docker.com/compose/networking/
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_voting
    #- CORE_LOGGING_LEVEL=ERROR
    - CORE_LOGGING_LEVEL=DEBUG
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_PROFILE_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
    - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
    - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
    - CORE_PEER_ID=peer0.Repubblicani.eurasia.com
    - CORE_PEER_ADDRESS=peer0.Repubblicani.eurasia.com:7051
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.Repubblicani.eurasia.com:7051
    - CORE_PEER_LOCALMSPID=RepubblicaniMSP
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
  command: peer node start
  # command: peer node start --peer-chaincodedev=true
  ports:
    - 7051:7051
    - 7053:7053
  volumes:
    - /var/run/:/host/var/run/
    - ../crypto-config/peerOrganizations/Repubblicani.eurasia.com/peers/peer0.
        Repubblicani.eurasia.com/msp:/etc/hyperledger/fabric/msp
    - ../crypto-config/peerOrganizations/Repubblicani.eurasia.com/peers/peer0.
        Repubblicani.eurasia.com/tls:/etc/hyperledger/fabric/tls
  depends_on:
    - orderer.eurasia.com
  networks:
    - voting

peer0.Monarchici.eurasia.com:
  container_name: peer0.Monarchici.eurasia.com
  image: hyperledger/fabric-peer
  environment:
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    # the following setting starts chaincode containers on the same
    # bridge network as the peers
    # https://docs.docker.com/compose/networking/
```

```
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_voting
      #- CORE_LOGGING_LEVEL=ERROR
      - CORE_LOGGING_LEVEL=DEBUG
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_GOSSIP_USELEADERELECTION=true
      - CORE_PEER_GOSSIP_ORGLEADER=false
      - CORE_PEER_PROFILE_ENABLED=true
      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
      - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
      - CORE_PEER_ID=peer0.Monarchici.eurasia.com
      - CORE_PEER_ADDRESS=peer0.Monarchici.eurasia.com:7051
      - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.Monarchici.eurasia.com:7051
      - CORE_PEER_LOCALMSPID=MonarchiciMSP
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: peer node start
    # command: peer node start --peer-chaincodedev=true
    ports:
      - 7051:7051
      - 7053:7053
    volumes:
      - /var/run/:/host/var/run/
      - ../crypto-config/peerOrganizations/Monarchici.eurasia.com/peers/peer0.
          Monarchici.eurasia.com/msp:/etc/hyperledger/fabric/msp
      - ../crypto-config/peerOrganizations/Monarchici.eurasia.com/peers/peer0.
          Monarchici.eurasia.com/tls:/etc/hyperledger/fabric/tls
    depends_on:
      - orderer.eurasia.com
    networks:
      - voting

  cli:
    container_name: cli
    image: hyperledger/fabric-tools
    tty: true
    environment:
      - GOPATH=/opt/gopath
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - CORE_LOGGING_LEVEL=DEBUG
      - CORE_PEER_ID=cli
      - CORE_PEER_ADDRESS=peer0.Stato.eurasia.com:7051
      - CORE_PEER_LOCALMSPID=StatoMSP
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/
          crypto/peerOrganizations/Stato.eurasia.com/peers/peer0.Stato.eurasia.com/
          tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/
          crypto/peerOrganizations/Stato.eurasia.com/peers/peer0.Stato.eurasia.com/
          tls/server.key
      - CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/
          peer/crypto/peerOrganizations/Stato.eurasia.com/peers/peer0.Stato.eurasia
          .com/tls/ca.crt
      - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/
          crypto/peerOrganizations/Stato.eurasia.com/users/Admin@Stato.eurasia.com/
          msp
      - CORE_CHAINCODE_KEEPALIVE=10
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: /bin/bash
    volumes:
      - /var/run/:/host/var/run/
      #- ./chaincode/:/opt/gopath/src/github.com/
      - ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
      - ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/
          channel-artifacts
    networks:
      - voting
    depends_on:
      - orderer.eurasia.com
      - peer0.Stato.eurasia.com
      - peer0.Repubblicani.eurasia.com
      - peer0.Monarchici.eurasia.com
```

Bibliografia

[1] Ferdinando M. Ametrano. Hayek money: the cryptocurrency price stability solution. Technical report, Milan Bicocca University, 2016. `https://papers.ssrn.com/abstract=2425270` [Online; acceduto 30/10/2017].

[2] Ferdinando M. Ametrano. Bitcoin and blockchain technology: An introduction, 2017. [Slides online, acceduto 02/11/2017].

[3] Saifedean Hisham Ammous. Blockchain technology: What is it good for? Technical report, Lebanese American University, 2016. `https://papers.ssrn.com/abstract=2832751` [Online; acceduto 02/11/2017].

[4] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timòn, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains, 2014. [Online, acceduto 30/10/2017].

[5] Imran Bashir. *Mastering Blockchain*. Packt, 2017.

[6] Vitalik Buterin. Casper version 1 implementation guide, 2017. [Online; acceduto 30/10/2017].

[7] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. Technical report, MIT, 1999. `http://pmg.csail.mit.edu/papers/osdi99.pdf` [Online; acceduto 23/10/2017].

[8] Manoj Debnath. Getting started with mongodb as a java nosql solution. [Online, acceduto 07/11/2017].

[9] Pedro Franco. *Understanding Bitcoin*. Wiley, 2015.

[10] Seth Gilbert and Nancy A. Lynch. Perspectives on the cap theorem. Technical report, MIT, 2002.

[11] Rick Echevarria (Intel). 2017: A summer of consensus, 2017. [Online; acceduto 30/10/2017].

[12] Repubblica Italiana. Costituzione, art. 48, 1948.

[13] Leslie Lamport, Marshall Pease, and Robert Shostak. The byzantine generals problem. Technical report, ACM, 1982. `http://lamport.azurewebsites.net/pubs/pubs.html#byz` [Online; acceduto 23/10/2017].

[14] David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus, 2016. [Online; acceduto 30/10/2017].

[15] DJ Qian. The fourth industrial revolution: Blockchain tech and the integration of trust, 2017. [Online, acceduto 02/11/2017].

[16] Jon-Mark Sabel. How AI is transforming pre-hire assessments - Hirevue blog, 2017. [Online; acceduto 23/10/2017].

[17] Melanie Swan. *Blockchain, Blueprint for a New Economy*. O'Reilly, 2015.

[18] Wikipedia. Ring signature. [Online, acceduto 06/11/2017].