UNIVERSITÀ POLITECNICA DELLE MARCHE

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

GESTIONE DELLE PRESCRIZIONI MEDICHE DEMATERIALIZZATE UTILIZZANDO UNA BLOCKCHAIN PRIVATA

MANAGEMENT OF DEMATERIALIZED MEDICAL PRESCRIPTIONS USING A PRIVATE BLOCKCHAIN

Relatore:
Prof. Luca SPALAZZI

Correlatore:
Prof. Marco BALDI

Tesi di laurea di:
Claudio PETROCCO

A. A. 2016/2017

Indice

Elenco delle figure

Elenco dei codici

Capitolo 1

Introduzione

In poco meno di vent'anni si è passati dal floppy disk al cd e, da questo, alla chiave usb e ad altri tipi di supporti portatili di memoria, fino ad arrivare al cloud computing, eliminando di fatto la necessità del supporto fisico per produrre e preservare i dati, le informazioni e i documenti digitali. L'affermarsi continua di nuove tecnologie ha quindi modificato la concezione del *documento*, in particolar modo della sua creazione e conservazione.
Se fino agli anni '90 si aveva una visione più ristretta e univoca del document management, che sostanzialmente coincideva con la digitalizzazione di immagini e l'archiviazione ottica di documenti nati originariamente su supporti analogici (prevalentemente cartacei), negli anni 2000 si è passati al concetto già più complesso di *Gestione documentale e conservazione sostitutiva* ovvero la sostituzione del documento cartaceo con l'equivalente documento digitale. Infatti con la Dematerializzazione il contenuto del documento viene "cristallizzato" grazie all'utilizzo della firma digitale e della marca temporale[1]. Oggi, invece, ci confrontiamo ormai con processi sempre più nativamente digitali dove non c'è più traccia dell'immagine della carta. D'altro canto, l'abbandono di supporti "analogici" in favore di strumenti digitali ha portato l'utente e i suoi documenti ad essere oggetto di numerose trappole in più. Se è vero che le tecnologie di conservazione e digitalizzazione hanno portato notevoli risparmi sia a livello di risorse utilizzate come carta, inchiosto e ore lavoro delle persone davanti alle stampanti, sia a livello di produttività personale essendo i documenti digitali disponibili istantaneamente è altresì vero che il rischio della falsificazione di documenti è un problema tutt'altro che remoto. Per poter, quindi, gestire correttamente queste nuove tipologie di documenti e informazioni rilevanti bisogna adottare specifici modelli e metodologie finalizzati a garantire l'attribuibilità, l'integrità, l'autenticità e la sicurezza nel tempo del complesso dei documenti digitalizzati.

[1] Garantendone così la sopravvivenza nel tempo come originale "autenticamente" digitale attraverso un sistema di conservazione a norma

Questa spinta verso nuovi processi digitali ha portato il Governo ad interessarsi verso la documentazione digitale arrivando ad emanare il decreto legislativo 82/2005 riguardante il *Codice dell'Amministrazione digitale*[1] che sancisce la definizione di:

- *documento analogico*: rappresentazione non informatica di atti, fatti o dati giuridicamente rilevanti.
- *documento informatico*: rappresentazione informatica di atti, fatti o dati giuridicamente rilevanti. Viene inquadrato come elemento centrale di quel processi di innovazione finalizzati alla completa digitalizzazione delle pratiche amministrative e viene dato pieno valore giuridico al processo di dematerializzazione.

Infine il Governo, tramite l'Agenzia per l'Italia Digitale (AgID), pubblica un white-paper riguardante la dematerializzazione della documentazione tramite supporti digitali.[2]

Obiettivi della tesi

In questo lavoro di tesi viene proposto un approccio alla digitalizzazione e alla conservazione di un documento sensibile quale la *ricetta medica*, basato sull'utilizzo della Blockchain, una tecnologia affermatasi recentemente e che sta riscuotendo notevole successo nei più svariati campi applicativi, dallo scambio di moneta digitale alla stipulazione di contratti, dalla gestione di dati sanitari alla pubblica amministrazione. L'avvento di questa tecnologia sta provocando notevole hype ed ottimismo e viene celebrata come una vera e propria rivoluzione tecnologia.
Il prototipo di applicazione per la gestione delle ricette bianche dematerializzate è basato sull'utilizzo di una determinata tipologia di blockchain, quelle private ed offre le seguenti caratteristiche:

- Meccanismi di garanzia di autenticità e validità delle ricette mediche dematerializzate;
- Elevata affidabilità e tolleranza ai guasti del sistema;
- Accorgimenti specifici per il trattamento dei dati (sensibili).

Struttura

Il testo della tesi è così strutturato:

Nel primo capitolo viene delineato lo stato dell'arte della sanità digitale in Italia. Viene descritto il Fascicolo Sanitario Elettronico (FSE) e la sua composizione, per poi passare alla ricettazione elettronica (che consiste in una delle principali fonti di dati per il FSE). Viene infine descritto il sistema di ricettazione elettronica delle Regione Marchie come caso di studio da cui partire per superare le criticità del sistema attuale.

Il secondo capitolo viene descritta la tecnologia che si è scelto di utilizzare nel presente lavoro di tesi ovvero la Blockchain. In particolare si illustra il suo funzionamento generale prima di passare alla descrizione della particolare tipologia di blockchain utilizzata nel presente lavoro di Tesi. Infine vengono descritte le attuali limitazioni della tecnologia per l'implementazioni di applicazioni decentralizzate complesse o con particolari requisiti di sicurezza.

Nel terzo capitolo viene descritta le "blockchain di seconda generazione" basata sull'utilizzo degli Smart Contract. Viene quindi descritto il funzionamento della blockchain Ethereum, contrapposta alla blockchain "classica", facendo riferimento alle potenzialità degli Smart Contract per la realizzazione di applicazioni decentralizzate complesse.

Nel quarto capitolo viene descritta la blockchain privata basata su Ethereum (e sugli Smart Contract) utilizzata nella progettazione dell'applicazione decentralizzata. Il capitolo mostra lo studio realizzato su questo software Ethereum based, denominato "Quorum" in relazione alle caratteristiche di sicurezza e privacy che va ad aggiungere rispetto alle blockchain pubbliche come Ethereum

Nel quinto capitolo vengono descritti tutti gli strumenti studiati ed utilizzati nell'implementazione dell'applicazione decentralizzata. Gli strumenti riguardano il linguaggio di programmazione orientato ai contratti ed i framework utilizzati per lo sviluppo.

Nel sesto capitolo viene descritto il procedimento di configurazione dell'ambiente di sviluppo. Vengono mostrati i passi per configurare i framework utilizzati che sono stati descritti nel capitolo precedente e viene anche mostrato il procedimento utilizzato per configurare correttamente un'istanza della blockchain Quorum per utilizzarla nell'implementazione

Nel settimo capitolo viene descritta l'architettura dell'applicazione e le funzionalità offerte.

Nel settimo capitolo vengono riassunte le considerazioni a livello di sicurezza dell'applicazione e le scelte prese nell'ambito dello sviluppo e dell'implementazione e le conclusioni.

Capitolo 2

Stato dell'Arte Sanità Digitale

1 Introduzione

Con *"sanità digitale"* si intendono gli interventi condivisi da tutte le Amministrazioni operanti a livello centrale, regionale e locale per quanto riguarda:

1. la digitalizzazione del ciclo prescrittivo;
2. la realizzazione di una soluzione federata di Fascicolo Sanitario Elettronico del cittadino;
3. l'aumento del tasso di innovazione digitale nelle aziende sanitarie;

In particolare, il *Fascicolo Sanitario Elettronico* (FSE) è l'insieme dei dati e documenti digitali di tipo sanitario e socio-sanitario generati da eventi clinici presenti e trascorsi, riguardanti l'assistito. Ha un orizzonte temporale che copre l'intera vita del paziente ed è alimentato in maniera continuativa dai soggetti che lo prendono in cura nell'ambito del SSN[1] e dei servizi socio-sanitari regionali ed è costituito, previo consenso dell'assistito, dalle Regioni e Province Autonome per le finalità di prevenzione, diagnosi, cura e riabilitazione perseguite dai soggetti del SSN e dei servizi sociosanitari regionali che prendono in cura l'assistito. È un investimento regionale, con una piattaforma FSE sovra-regionale. La normativa che regola la struttura del fascicolo è contenuta nel Decreto Legislativo 18/2012[3] convertita, con modificazioni, dalla legge 17 dicembre 2012 del Decreto Legislativo 69/2013[4]. La *Tessera sanitaria (TS)* invece, istituita ai sensi dell'articolo 50, comma 1, del decreto legge 269/2003, abilita all'accesso delle prestazioni sanitarie erogate dal SSN su tutto il territorio nazionale ed è Tessera di assicurazione malattia ai fini del riconoscimento dell'assistenza sanitaria nei Paesi UE, oltre a fungere da codice fiscale.

[1] Sistema Sanitario Nazionale

Infine, per quanto riguarda le *Ricette digitali*, l'art. 50 della Legge 326/2003 (modificato dalla Legge finanziaria 2007) ha introdotto l'obbligo di trasmissione telematica dei dati delle ricette ai fini del controllo della spesa, ed il DL 78/2010 (art 11, comma 16) ha dato valore legale alla trasmissione telematica dei dati delle ricette (scompare "ricetta rossa" cartacea).
Tutto il lavoro riguardante la sanità digitale in Italia è stato portato avanti dal gruppo di lavoro coordinato dall'AgID[2], il quale ha rilasciato le Specifiche tecniche per l'interoperabilità tra i sistemi regionali del Fascicolo Sanitario Elettronico, comprendente il framework e dataset dei servizi base. Questo lavoro è stato il risultato dei test effettuati dalla Regione Emilia-Romagna, Lombardia e Veneto che, su proposta AgID, si sono offerte, con il supporto del CNR, di validare le specifiche di dettaglio per l'interoperabilità dei sistemi regionali di FSE ed in particolare i servizi di ricerca, recupero e indicizzazione dei documenti che compongono il Fascicolo.
Durante questa fase di progettazione inoltre, si è pronunciato anche il Garante della privacy [5] a tutela della protezione dei dati personali dei pazienti, per quanto riguarda il fascicolo sanitario elettronica e la refertazione online. Infatti, il garante ha deliberato che "il paziente deve poter scegliere, in piena libertà, se far costituire o meno un fascicolo sanitario elettronico, con tutte o solo alcune delle informazioni sanitarie che lo riguardano; deve poter manifestare un consenso autonomo e specifico, distinto da quello che si presta a fini di cura della salute; al paziente deve essere inoltre garantita la possibilità di "oscurare" la visibilità di alcuni eventi clinici. Per poter esprimere scelte consapevoli il paziente deve essere adeguatamente informato. Con un linguaggio comprensibile e dettagliato l'informativa deve quindi indicare chi (medici di base, del reparto ove è ricoverato, farmacisti) ha accesso ai suoi dati e che tipo di operazioni può compiere. Il fascicolo sanitario elettronico potrà essere consultato dal paziente con modalità adeguate (ad es. tramite smart card) e dal personale sanitario strettamente autorizzato, solo per finalità sanitarie. Non potranno accedervi invece periti, compagnie di assicurazione, datori di lavoro.

2 Fascicolo Sanitario Elettronico

La realizzazione del Fascicolo Sanitario Elettronico in Italia ha visto, nel corso degli ultimi anni, uno sviluppo differenziato sul territorio nazionale a seconda dei contesti Regionali e Provinciali, nell'ambito dei quali alcune Amministrazioni hanno realizzato o avviato la realizzazione di infrastrutture di Fascicolo, mentre altre

[2] Agenzia per l'Italia Digitale

hanno sviluppato esperienze pilota significative.
In questo contesto le recenti modifiche al D.L. n°179/2012 operate dal D.L. 69/2013[4] e dalla successiva L. di conv. n°98/2013, hanno dato un impulso decisivo alla realizzazione del Fascicolo Sanitario Elettronico sul territorio nazionale.[3]
Le normative hanno inquadrato la fase implementativa nel seguente approccio:

1. emanazione dello schema di DPCM attuativo per l'FSE;[6]
2. emanazione, a cura dell'Agenzia per l'Italia Digitale e del Ministero della Salute, di Linee Guida dettagliate per il progetto del Fascicolo Sanitario Elettronico a partire dallo standard HL-7 EHR-S;
3. presentazione del piano di progetto per la realizzazione del FSE da parte di ciascuna Regione e Provincia Autonoma.

Nelle specifiche tecniche del fascicolo sanitario elettronico, scritte dall'AgID, si afferma che Il nucleo minimo del fascicolo, uguale per tutti i fascicoli istituiti da Regioni e Province autonome, è costituito dai seguenti dati e documenti:

1. dati identificativi e amministrativi dell'assistito;
2. referti;
3. verbali pronto soccorso;
4. lettere di dimissione;
5. profilo sanitario sintetico;
6. dossier farmaceutico;
7. consenso o diniego alla donazione degli organi e tessuti;

Il risultato più importante della fase di progettazione è stato la definizione di un modello funzionale del FSE[4] al fine di evitare la proliferazione di sistemi funzionalmente incompatibili. Questo perchè le funzioni da realizzare devono essere conformi ad un modello funzionale di riferimento condiviso su scala nazionale e reimplementato a livello di singola regione.
Il modello così progettato, è diviso in due blocchi funzionali principali:

- *Servizi di interfaccia*: consentono l'interazione da parte degli attori e delle componenti esterne con il sistema di FSE regionale. Ci sono ad esempio i servizi di interfaccia interregionale;

[3] obbligatoria l'istituzione in tutte le Regioni italiane entro il 30 giugno 2015
[4] ottenuto dalla localizzazione italiana dello standard ISO/HL7 EHR-S FM R2

- *Macro funzioni*: raccolgono le funzioni del modello logicamente correlate. Tra queste funzioni troviamo ad esempio le funzioni di indicizzazione dei documenti, quelle di alimentazione, quelle per la sicurezza e la privacy etc.

La struttura generale infatti è la seguente:

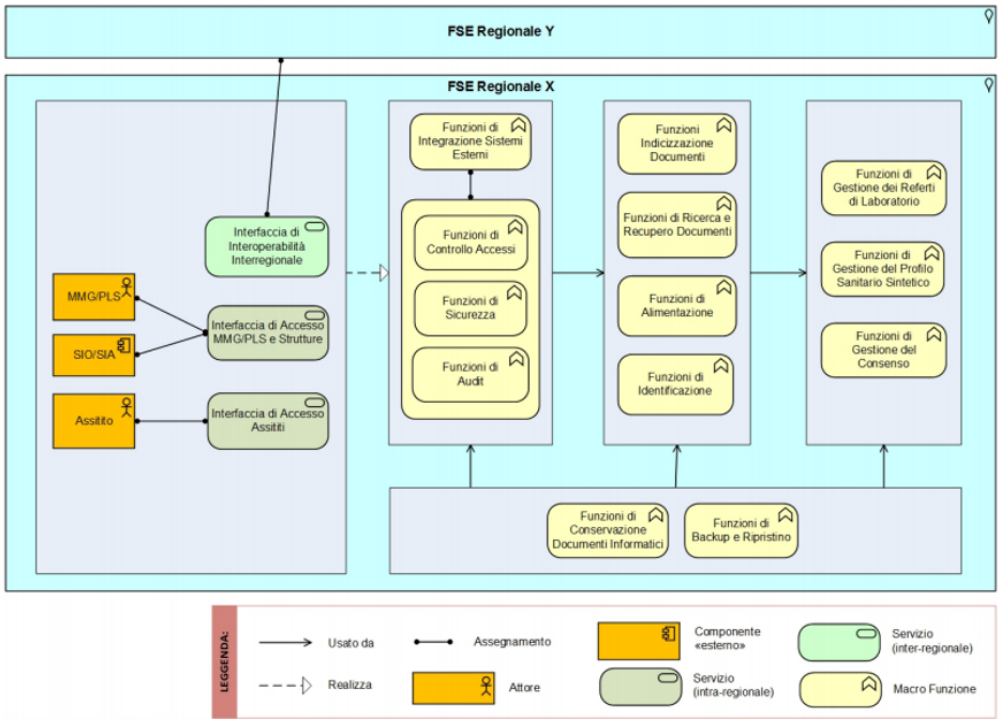

Figura 2.1: Modello funzionale FSE

Mentre le principali fonti di informazioni che il Fascicolo Sanitario Elettronico raccoglie sono le seguenti:

La realizzazione di sistemi di FSE così strutturati costituisce, a livello regionale, un salto di notevole importanza per il miglioramento dell'efficacia ed efficienza delle cure percepite dai cittadini all'interno di un sistema sanitario regionale rispetto al modello precedente basato su registri e cartelle cartacee e blocchi di prescrizione stampati dalla zecca di Stato e consegnati personalmente ad ogni medico del distretto sanitario di appartenenza.
Una delle fonti di informazioni maggiori del Fascicolo Sanitario Elettronico, che è anche oggetto di studio in questo lavoro di tesi, risulta essere composta dall'insieme delle prescrizioni che i medici generano per i propri pazienti e che vengono erogate dalle farmacie presenti sul territorio nazionale. Il governo ha inoltre lasciato libertà

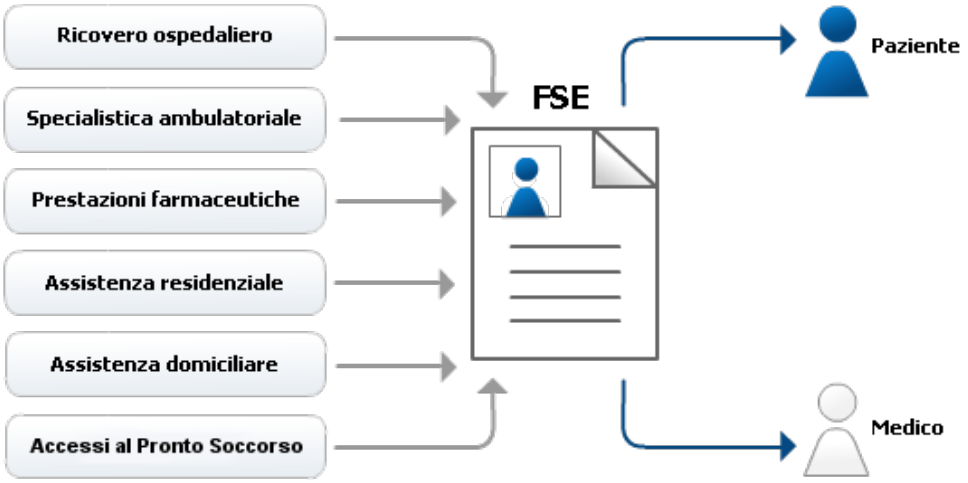

Figura 2.2: Schema dei dati raccolti dal FSE

di implementazione e gestione delle strutture alle singole regioni portando così alla creazione di sistemi eterogenei che fanno capo ad alcuni punti di riferimento gestiti dal Governo tramite le infrastrutture della Sogei. Il sistema complessivo si presenta come un sistema di tipo centralizzato caratterizzato dalle tipiche problematiche dei central points of failure. Inoltre, per quanto riguarda il sistema delle prescrizioni, esso gestisce solo le ricette mediche *rosse* mentre le ricette mediche bianche non sono oggette ancora di prescrizione elettronica in tutte le regioni. Le funzionalità da cui si è partito per sviluppare questo lavoro di Tesi sono quelle riguardanti la prescrizione e l'erogazione delle ricette mediche *bianche* che differiscono dalle ricette mediche rosse per una serie di caratteristiche che saranno illustrate nel proseguio del capitolo.

3 Ricetta Elettronica

Parallelamente al processo di istituzione del Fascicolo Sanitario Elettronico, anche il sistema di prescrizione farmaceutica è, al momento, oggetto di grandi modifiche in quanto sta diventando digitale.
Infatti, attraverso il *Sistema Tessera Sanitaria* (TS), è stata realizzata la diffusione della ricetta elettronica, con la Dematerializzazione delle ricette mediche, introdotta con il D.L. n°78/2010, il decreto ministeriale del 2/11/2011 ed il D.L. n°179/2012. [7] Questo sistema è il sistema informativo centralizzato, realizzato in attuazione dell'art 50 del D.L. 269/2003, dal Ministero dell'economia e delle finanze - Ragioneria Generale dello Stato e gestito per la parte informatica da Sogei[5]. La ricetta elettronica prevede la completa eliminazione del supporto cartaceo della ricetta

[5]Società di Information Technology interamente controllata dal Ministero dell'Economia e delle Finanze

nell'intero iter che va dalla fase di prescrizione del medico, alla erogazione della prestazione, al successivo controllo e rendicontazione.
Le finalità del progetto sono:

- potenziamento dei controlli e prevenzione errori nelle prescrizioni;
- maggiore interscambio di informazioni favorito dal formato digitale;
- semplificazioni accesso prestazioni sanitarie a carico del SSN;

La prescrizione elettronica quindi, si presenta come il primo importante tassello nella costruzione del FSE in quanto rappresenta una delle porzioni più importanti dei dati clinici del paziente.
Esistono due tipi principali di ricette:

- *Ricetta rossa*: può essere compilata solamente dai medici dipendenti di strutture pubbliche o convenzionati con il servizio sanitario nazionale e viene utilizzata per la prescrizione di una terapia farmacologica, la prescrizione di un esame diagnostico o una visita specialistica a carico del servizio sanitario. L'uso di una ricetta rossa non permette l'erogazione a carico del servizio sanitario di farmaci o prodotti parafarmaceutici non compresi tra le formulazioni del prontuario farmaceutico regionale, né di esami, visite o terapie non comprese nei Lea o nelle disposizioni della propria regione. Un esempio è illustrato nella seguente figura:

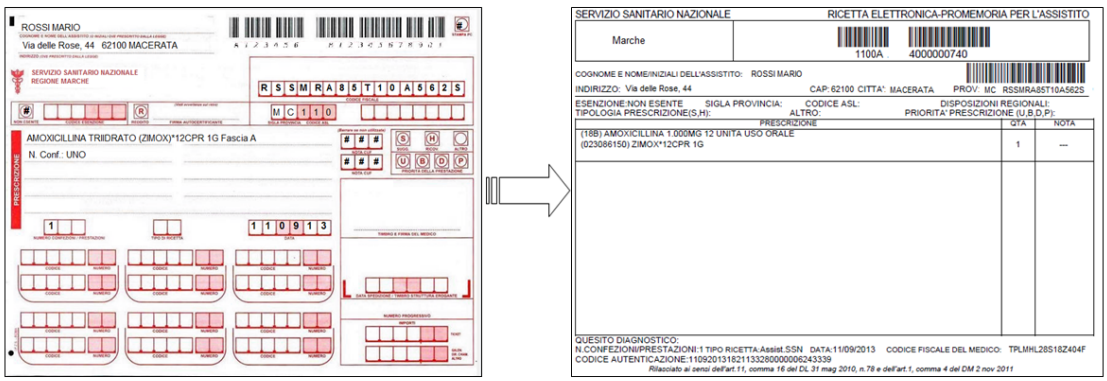

Figura 2.3: Ricetta rossa dematerializzata

- *Ricetta bianca*: ricetta che il medico compila su carta bianca, sulla quale siano però riportati: il nome e cognome del medico; la data; il luogo; la firma autografa del medico. In questo caso, il nome dell'assistito non è strettamente

necessario. Su ricetta bianca possono essere prescritte tutte le prestazioni di specialistica ambulatoriale, di diagnostica strumentale e di laboratorio, di norma correlate alla propria branca di specializzazione e i farmaci, prestazioni che saranno sempre a carico del cittadino assistito. Per la prescrizione a carico del servizio sanitario è infatti necessaria la ricetta del ricettario regionale ed è valida in tutte le farmacie italiane. Una delle attuali criticità del sistema è che il procedimento della dematerializzazione delle ricette non si applica per le ricette bianche che vengono ancora compilate su normale carta:

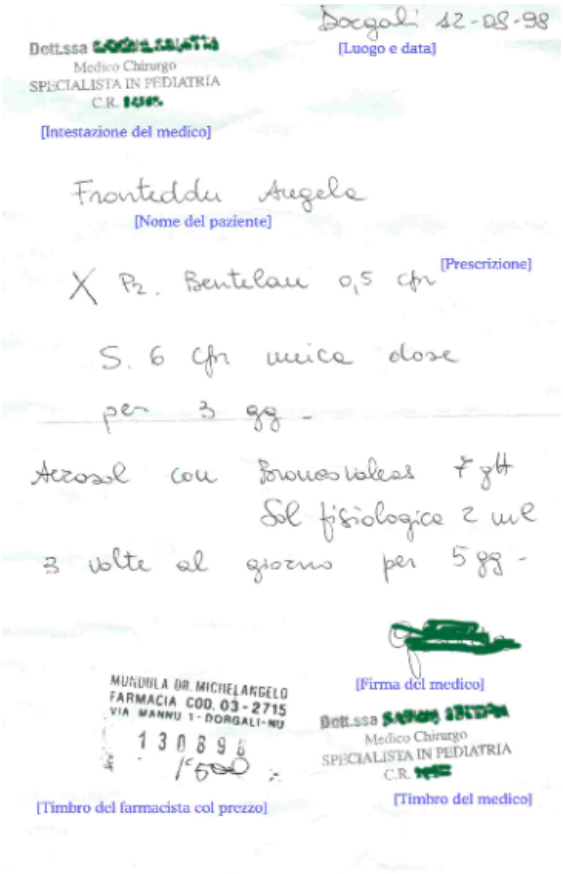

Figura 2.4: Ricetta bianca

4 Architettura del Sistema

L'attività di accoglienza dei dati delle ricette prescritte, prevede l'utilizzo di due sistemi centrali:

1. *Sistema di accoglienza* (SAC): è il Sistema di Accoglienza Centrale gestito da Sogei per il Ministero dell'Economia e delle Finanze (MEF). Il SAC gestisce

la circolarità nazionale delle ricette dematerializzate, ovvero la possibilità per gli assistiti di utilizzare una prescrizione in qualunque Regione, garantendo anche che la stessa prestazione non sia erogata più volte in regioni diverse;

2. *Sistema di accoglienza regionale* (SAR): è il Sistema di Accoglienza Regionale gestito da CUP 2000 che colloquia con il Sistema di Accoglienza Centrale del Ministero (SAC) per l'invio e il recupero delle ricette dematerializzate, ed aggiorna lo stato della ricetta (prenotata, sospesa,bloccata ed erogata) E' in grado di:

   - raccogliere tutte le prescrizioni dematerializzate inviate dai medici prescrittori della Regione;
   - rendere disponibili le prescrizioni dematerializzate ai sistemi che, se abilitati, possono accedere alla ricetta stessa (CUP, accettazioni, farmacie, etc.);
   - fornire servizi di ritorno informativo verso le Aziende sanitarie e le farmacie;
   - sviluppare le regole e i controlli regionali necessari a garantire l'emissione di prescrizioni corrette;

Questi due sistemi coesistono contemporaneamente, in quanto nel caso in cui le regioni non avessero sviluppato un proprio sistema di accoglienza, il SAC si configura come sistema di riferimento per i medici prescrittori, le farmacie e le strutture di erogazione di prestazioni di specialistica ambulatoriale, generando per questi le credenziali di accesso al sistema[6]. Invece nel caso in cui la regione abbia a disposizione un SAR proprio, esso si andrà ad interfacciare con il SAC, che fungerà da secondo centro di accoglienza con le stesse funzionalità del SAR, solamente estese a livello nazionale.

4.1 Caso di studio: Regione Marche

Nella Regione Marche, la delibera 677[8] della Giunta Regionale 04/06/2014 per l'approvazione dello "schema di protocollo di intesa con i medici di medicina generale per la riqualificazione della medicina del territorio e la messa a regime della rete regionale per la ricetta dematerializzata e per l'implementazione dei flussi di dati" che ha dato il via al progetto di dematerializzazione delle ricette SSN il quale prevede la sostituzione della ricetta rossa cartacea con la ricetta elettronica dematerializzata. La procedura per l'emissione di una ricetta è la seguente:

[6]abilitazione al SAC. Credenziali inviate tramite busta chiusa alle ASL territoriali

- il medico prescrittore:

    1. si collega al Sistema Centrale Tessera Sanitaria (anche attraverso un eventuale SAR)[7];

    2. genera i dati della ricetta elettronica. Il contenuto della ricetta viene salvato nel Sistema Tessera Sanitaria a livello centrale (tenuto sempre conto che il contenuto sarà anche salvato nel SAR, se presente). La ricetta sarà identificata tramite un codice univoco nazionale denominato *Numero di Ricetta Elettronica* (NRE) e conterrà i seguenti elementi: NRE, dati anagrafici dell'assistito titolare della prescrizione, eventuali esenzioni e la prestazione da erogare;

    3. se i dati risultano corretti, il medico rilascia al paziente un *promemoria* cartaceo contenente i dati della ricetta dematerializzata;

- la struttura erogatrice:

    1. presso la struttura erogatrice, l'assistito presenterà il promemoria;

    2. la struttura si collegherà al sistema centrale (SAR o SAC) e procederà alla ricerca della ricetta attraverso l'NRE riportato su di essa, unitamente al codice fiscale dell'assistito riportato sulla sua tessera sanitaria;

    3. nel caso in cui la ricetta risulti erogabile, la struttura procede all'erogazione della prestazione comunicando al tempo stesso le informazioni sulla prestazione erogata (al SAR e/o al SAC);

    4. completata l'erogazione la struttura ritira il promemoria al paziente;

Questa sequenza viene mostrata nella figura seguente:

[7] essendo nelle Marche presente anche il SAR, la comunicazione avviene verso il SAR e poi verso il SAC e successivamente viene effettuato un controllo incrociato delle informazioni inserite nei due sistemi

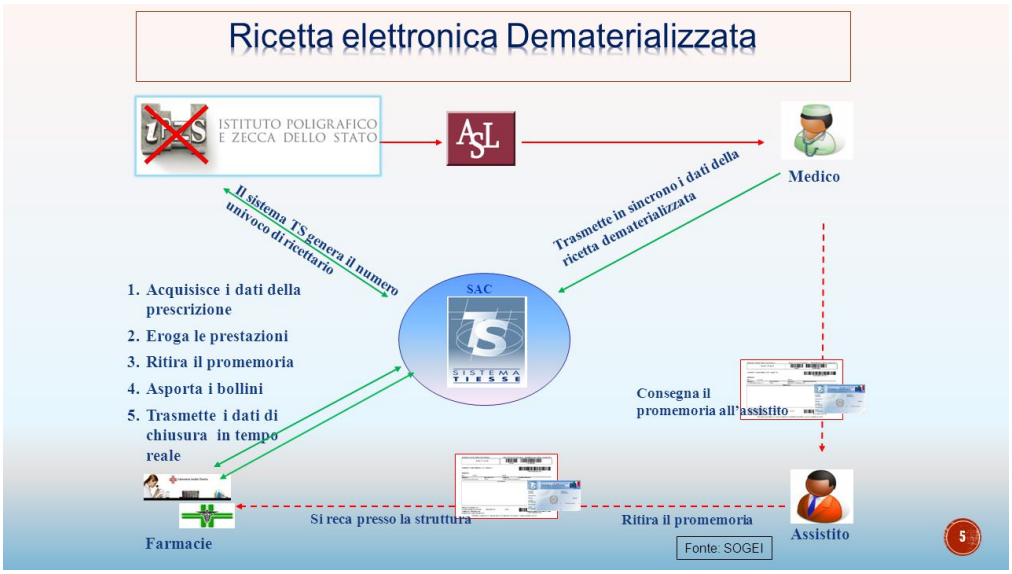

Figura 2.5: Diagramma dell'erogazione di una ricetta

Il complesso dei servizi offerti dal FSE nell'ambito della Regione Marche è garantito da un insieme di componenti applicative cooperanti tra loro supportate da un apparato infrastrutturale allocato presso il data center sanità e i nodi aziendali, il tutto integrato all'interno di un contesto regionale caratterizzato dall'esistenza di sistemi preesistenti, con cui il FSE si troverà ad interagire per l'esposizione dei propri servizi rispetto al SAC centrale. Le componenti sono eterogenee e sono riassunte nella seguente figura:

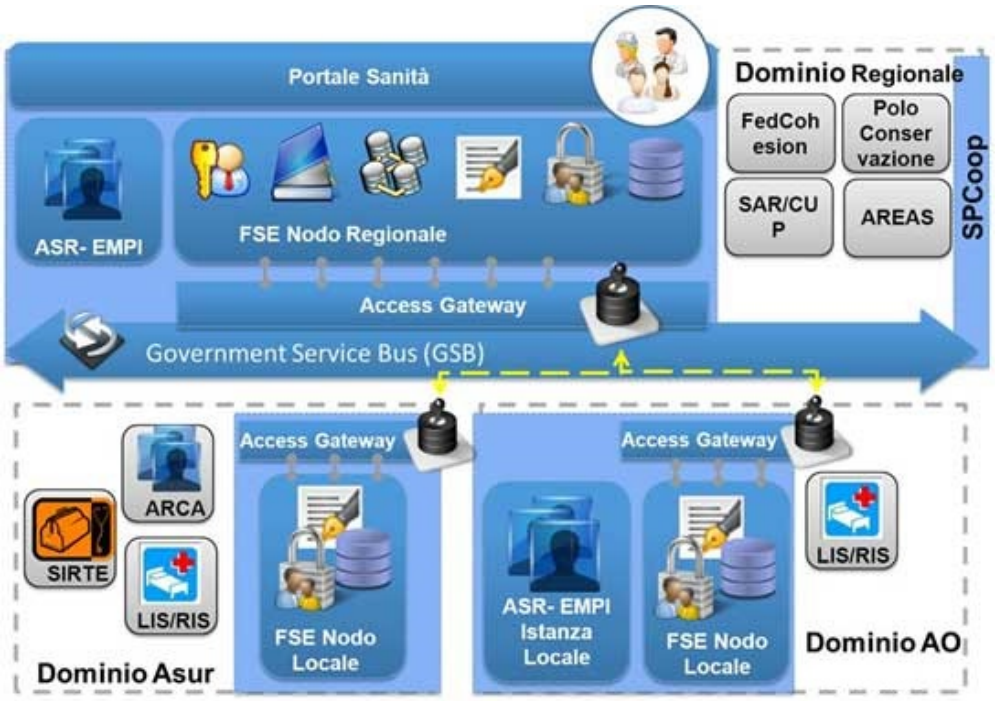

Figura 2.6: Schema architetturale sanità digitale Marche

5 Una soluzione decentralizzata

5.1 Criticità del sistema

Questo sistema ha dimostrato alcune criticità a seguito delle sperimentazioni fatte e dell'attuale utilizzo:

- la mancanza di collegamenti internet o di una piattaforma di gestione troppo lenta in alcune zone crea problemi di utilizzo e di sovraccario del sisstema;
- la presenza di due distinti sistemi di trasmissione dati e accoglienza delle ricette (SAC/SAR) ha aggiunto un ulteriore grado di complessità alle architetture dei sistemi a livello regionale ed inter-regionale in quanto alcune regioni hanno optato per l'utilizzo del SAC mentre altre del SAR (che si interfaccia con il SAC). Inoltre, normalmente un SAR risulta essere più lento di un SAC a livello di gestione del flusso dei dati;
- la prescrizione delle ricette bianche risulta essere esclusa dalla dematerializzazione. Quindi una ricetta bianca può essere oggetto di falsificazione o truffa, nel caso in cui il blocco di ricette con i dati del medico venga rubato;

Inoltre, possono verificarsi casi in cui i server di Sogei del SAC responsabili della prescrizione e spedizione della ricetta digitale risultino irragiungibili con conseguenze che vanno dal mancato invio della ricetta digitale al non poter verificare la veridicità del contenuto di un promemoria se non in modo "postumo" all'erogazione del contenuto della prescrizione cartacea.

5.2 La decentralizzazione

Una possibile soluzione di queste criticità può essere trovata nell'utilizzo di un approccio decentralizzato che risiede nell'utilizzo della blockchain (si eliminano i Single Point of Failure) per registrare gli eventi (che saranno organizzati in blocchi) e le transazioni di prescrizione ed erogazione. L'utilizzo della blockchain ci garantisce:

- *Non ripudio*: la decisione riguardante la validità di un'informazione non viene presa unilateralmente ma attraverso un meccanismo di raccolta del consenso all'interno della rete, rendendo così particolarmente difficile metterne in discussione l'esito;
- *Autenticità*: tutti gli eventi possono essere fatti risalire con certezza alle identità digitali che li hanno generati, attraverso l'utilizzo di meccanismi di crittografia asimmetrici;

- *Integrità*: i dati che sono stati scritti all'interno della Blockchain non possono essere modificati, se non attraverso specifiche regole del protocollo che definiscono rigorosamente le modalità con cui si possono effettuare cambiamenti;
- *Tracciabilità*: a tutti gli eventi registrati vengono assegnati un identificativo e una marca temporale che li rende facilmente tracciabili e verificabili.
- *Programmabilità*: all'interno dei blocchi possono essere incluse istruzioni che facciano scatenare specifiche azioni al verificarsi certe condizioni.

Capitolo 3

Blockchain

1 Introduzione

Una blockchain è una base di dati distribuita organizzata in blocchi, ciascuno dei quali contiene un insieme di record. Questi ultimi rappresentano la registrazione di un particolare evento associato ad un istante temporale (un timestamp). La struttura complessiva della rete assume la forma di una catena poichè ciascun blocco è legato al precedente, come è illustrato nella seguente immagine:

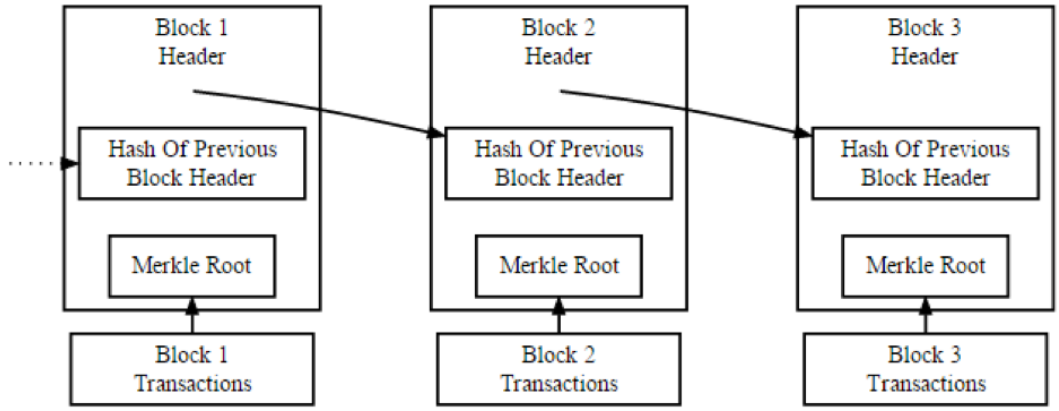

Figura 3.1: Catena di blocchi in una blockchain

Il legame tra un blocco ed il suo predecessore viene realizzato attraverso l'inserimento, in ciascun blocco, di un riferimento al blocco precedente. Questa dipendenza all'indietro trova la sua conclusione nel blocco iniziale della catena, che solitamente viene generata da zero. Questa tecnologia nasce come infrastruttura della criptovaluta conosciuta come *Bitcoin*, il cui fondatore, noto con lo pseudonimo di Satoshi

Nakamoto, l'ha creata nel 2009 generando il primo blocco dopo aver spiegato il funziomento del sistema nel suo documento pubblicato un anno prima. [9]
Ma questa non è l'unica definizione che si può dare della blockchain in quanto ogni definizione può mettere in evidenza uno o più aspetti salienti della tecnologia stessa.

Blockchain come evoluzione del concetto di "Ledger" Con *Ledger* si intende il "libro mastro" in cui vengono registrate tutte le transazioni del sistema. Sotto quest'ottica la Blockchain è la realizzazione del *Distributed Ledger*, come evoluzione dal *Centralized Ledger*, passando per il *Decentralized Ledger*. Tipicamente, la logica centralizzata è rappresentata dal tradizionale "Centralized Ledger" con un rapporto rigorosamente centralizzato "Uno-A-Tanti", dove tutto deve essere gestito facendo riferimento a una struttura o autorità o sistema centralizzato. Nel Centralized Ledger la fiducia è nell'autorità, nell'autorevolezza del soggetto o sistema che rappresenta il "centro" dell'organizzazione.
Dal modello centralizzato, si passa attraverso il "Decentralized Ledger", che ripropone la logica della centralizzazione a livello "locale" con "satelliti" organizzati a loro volta nella forma di Uno-A-Tanti che si relazionano a loro volta in una forma che ripete il modello Uno-A-Tanti. Non c'è più un "grande" soggetto centrale ma ci sono tanti "soggetti centrali". La fiducia anche in questo caso viene delegata a un soggetto centrale, logicamente più vicino, ma comunque centralizzato. Le organizzazioni basate su Decentralized Ledger definiscono una Governance che stabilisce delle forme di coordinamento di tipo centralizzato.
Infine si arriva al "Distributed Ledger", ovvero ad una reale e completa logica distribuita dove non esiste più nessun centro e dove la logica di governance viene costruita attorno a un nuovo concetto di fiducia tra tutti i soggetti. Il processo decisionale passa attraverso un rigoroso processo di costruzione del Consenso.

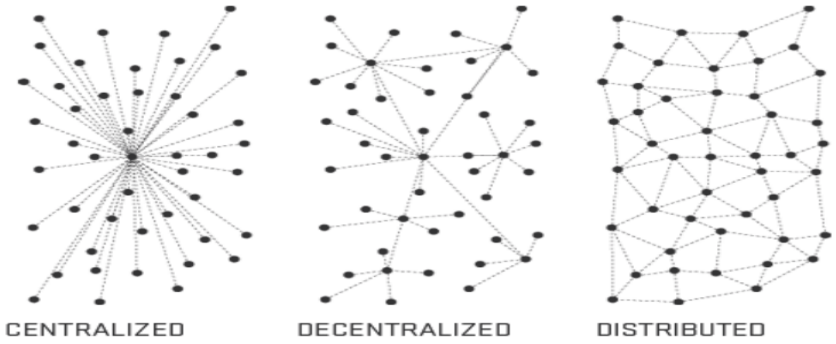

Figura 3.2: Da Centralized a Decentralized Ledger

2 Caratteristiche Generali

La blockchain si evolve attraverso l'attività di un timestamp server distribuito, che prende un blocco di item e tramite la Marca Temporale[1] marca il blocco e ne calcola l'hash. Mentre nei centralized ledger il punto di forza era localizzato nella fiducia che tutti i partecipanti dovevano avere nel gestore del central ledger (un soggetto di terze parti al di sopra dei partecipanti che garantisce la fiducia), nel distributed ledger la blockchain può essere considerata come una base di dati senza intermediari, ovvero che per utilizzarla non è necessario rivolgersi ad un server. Infatti le sue caratteristiche e la sua natura decentralizzata sono quelle che garantiscono l'affidabilità e la fiducia senza la presenza di un'autorità centrale che regoli le attività dei vari utenti. Questo ad esempio, nel caso di Bitcoin, si traduce nell'eliminazione di entità bancarie in grado di controllare (ed eventualmente anche alterare) le transazioni.
Una blockchain gode quindi delle seguenti proprietà:

1. è distribuita: le informazioni sono replicate e dislocate su più nodi della rete;

2. è inviolabile: essendo i blocchi condivisi tra più nodi ed dato che, per alterarne il contenuto è necessario il consenso della maggioranza assoluta, è improbabile che un attacco vada a buon fine;

3. può essere pubblica (Unpermissioned Ledger) dove chiunque può controllare l'attività di chiunque, ma può anche essere privata (Permissioned Ledger) dove i partecipanti alla blockchain sono un numero limitato di attori che sono definiti come trusted. Le Permissioned Ledger sono più vicine alle esigenze di imprese ed istituzioni che vogliono comunque servirsi della tecnologia della blockchain;

4. rispetta i requisiti di:

   - *confidenzialità*: le informazioni sono accessibili solo agli utenti autorizzati;
   - *integrità*: ognuno può alterare solo i dati per cui è autorizzato;
   - *non ripudio*: le transazioni irrevocabili (caratteristica fondamentale della blockchain);
   - *autenticità*: vengono utilizzati meccanismi di cifratura per garantire l'autenticità nella blockchain.

[1] istante temporale o timestamping

3 Architettura

L'architettura di una blockchain (semplificata), può essere osservata nella seguente figura:

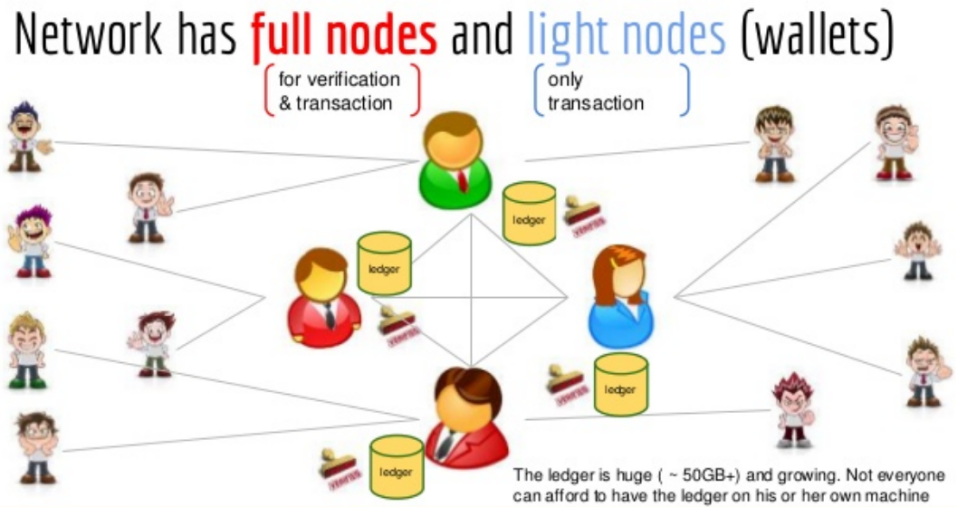

Figura 3.3: Architettura di una blockchain su cui operano full node e lightweight node

Possiamo notare come l'assenza di intermediari si rispecchi a livello architetturale. Infatti la rete su cui si basa questa tecnologia è di tipo "peer to peer". Questa architettura risulta essere scalabile, dato che ciascun nodo può essere rimosso dalla rete o aggiunto alla stessa senza che l'attività complessiva risulti compromessa. Un nodo, per unirsi alla blockchain, ha due possibilità:

1. diventare un *full node*: il nodo deve scaricare una copia della blockchain al fine di poter effettuare operazioni e, al tempo stesso, validare le operazioni effettuate dagli altri utenti della blockchain;

2. diventare un *lightweight node*: il nodo può comunque partecipare alle attività della blockchain, senza però validare le operazioni degli altri utenti. Il vantaggio di un lightweight node risiede nel fatto che non deve scaricare l'intera copia della blockchain (ne scarica solamente una parte) per poter compiere delle operazioni ma agisce interrogando esso stesso uno dei full node presenti nella rete.

4 Mining e Proof of Work

Il processo di consenso decentralizzato del Bitcoin richiede l'esistenza dei nodi nel network che continuamente tentano di produrre pacchetti di transazioni chiamati

"blocchi". Il network produce all'incirca un blocco ogni dieci minuti ed ogni blocco contiene, come abbiamo detto in precedenza, una marca temporale, un numero casuale(nonce), un riferimento al precedente blocco (l'hash del precedente blocco) e una lista di tutte le transazioni che sono avvenute dal precedente blocco. Con il passare del tempo si crea quindi una persistente, sempre crescente catena che si aggiorna costantemente per rappresentare l'ultimo stato del libro mastro di Bitcoin. Questo processo di creazione di un nuovo blocco va sotto il nome di *mining* e viene realizzato da entità facenti parte della rete chiamate *miners*. Per generare un blocco generalmente è necessario:

- stabilire quali transazioni andranno a far parte del nuovo blocco;
- verificare la validità di queste transazioni;
- selezionare il blocco più recente della blockchain (l'ultimo creato) ed inserire il suo hash nel nuovo blocco;
- risolvere infine la *Proof-of-work* (POW) e, contemporaneamente, assicurarsi che eventuali blocchi generati prima del proprio siano validi.

La componente principale dell'attività di mining è la fase che riguarda la Proof-of-work, ovvero quella che involve tutti i miners e che li vede impegnati nella risoluzione di un problema ad elevata complessità (in cui è richiesta un'elevata capacità di calcolo), la cui risoluzione avrà come risultato l'aggiunta di un nuovo blocco alla rete e un pagamento in Bitcoin al miner che ha risolto il blocco. La POW si articola nel seguente modo: viene utilizzato l'algoritmo di hash SHA-256 che produce un numero formato da 256 bit. L'oggetto su cui si applica la funzione di hash, costituito dal blocco di transazioni corrente, è il cosiddetto *header*[2] del blocco. L'obiettivo di questa sfida è ottenere un valore di hash inferiore di un target regolato dinamicamente[3]. Poichè SHA-256 è progettato peressere una funzione completamente imprevedibile e pseudo-randomica, l'unico modo per creare un blocco valido è un insieme di tentativi con l'incremento ripetuto del nonce al fine di trovare il valore richiesto dalla proof-of-work. Si nota come questa sfida computazionale sia difficile da completare ma facilmente verificabile all'interno della rete.
Una volta risolta la proof-of-work e generato il blocco associato, per alterare una transazione in un blocco, bisognerebbe modificare anche tutte le transazioni passate (essendo esse collegate in cascata da una serie di firme digitali) e rimettere quindi

[2]comprende il numero di versione del protocollo, l'hash del blocco precedente, l'hash di un sommario delle transazioni del blocco corrente, una timestamp (cioè data e ora di creazione del blocco) e il grado di difficoltà stabilito per il blocco corrente blocco e il nonce

[3]generalmente il target viene ricalibrato dal network ogni 2016 blocchi, al fine di lasciar costante il tempo di produzione di un blocco (10 minuti)

in gioco tutta la capacità di calcolo utilizzata nella risoluzione perchè si dovrebbe rimodificare quel blocco e tutti i suoi successori. Infine al minatore che ha risolto il blocco generato viene ricompensato con una certa quantità di bitcoin. In altre implementazioni della blockchain esistono altri strumenti per stabilire chi andrà a minare il blocco successivo:

- *proof-of-stake*: il diritti di minare viene concesso in funzione della quantità di denaro che si possiede (adatta in contesti con bassa capacità di calcolo);
- *prrof-of-burn*: il miner può aggiudicarsi la contesa dimostrando di aver consumato monete, inviandole ad un indirizzo non suo e non potendole quindi più utilizzare, dimostrando di aver fatto un sacrificio.

5 Le transazioni

Una transazione rappresenta un trasferimento di valore che viene registrato in un blocco della blockchain. Bitcoin, ad esempio, utilizza un sistema di transizioni di stato dove:

- stato: consistente nella proprietà dello status di tutti i bitcoins esistenti;
- funzione di transizione di stato: riceve uno stato ed una transizione e trasmette un nuovo stato che ne costituisce il risultato.

In bitcoin lo "stato" è la raccolta di tutte le monete (tecnicamente definite come "transazioni in uscita non spese" o UTXO[4]). Ogni UTXO ha una denominazione ed un proprietario definito da un indirizzo di 20 byte che corrisponde essenzialmente alla sua chiave pubblica. Una transazione viene costruita in riferimento a transazioni passate da cui prelevare i fondi che vengono trasferiti al nuovo destinatario. Quindi ogni transazione contiene:

- uno o più vettori di input, dove ogni input contiene un riferimento ad uno UTXO esistente ed una firma crittografica prodotta da una chiave privata associata all'indirizzo del proprietario[5];
- uno o più vettori di output, dove ogni output contiene un nuovo UTXO che deve essere aggiunto allo stato;

Una transazione può avvenire nel seguente modo:

[4]Unspend Transaction Output
[5]in questo modo si evita la possibilità che una persona spenda monete non di sua proprietà

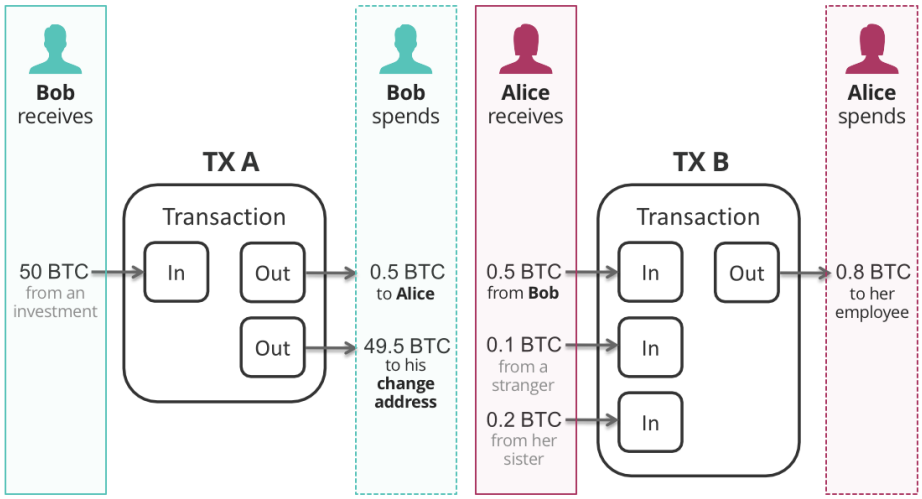

Figura 3.4: Esempio di transazione

Ogni transazione effettuata viene quindi inviata a tutti i nodi i quali raccolgono le nuove transazioni in un blocco e partecipano alla proof-of-work per minare il nuovo blocco. Un blocco, per essere accettato deve contenere al suo interno solamente transazioni valide e non deve essere presente il problema del *double spending* (spendere gli stessi soldi due volte).

6 Permissioned Blockchain

Sono attualmente presenti numerosi progetti di blockchain permissioned, come il progetto Hyperledger[10] della Linux Foundation, o la rete Gem Health[11]. Alcuni ritengono che le blockchain private siano sono un modo alternativo di definire uno shared-database. Infatti, come sostenuto da Gideon Greenspan, CEO di CoinSciences :"se fiducia e robustezza non sono un target, non c'è nulla che una blockchain possa fare ed un database no", a conferma del fatto che le due tecnologie sono tutto sommato alternative, ma non analoghe.
Nel presente lavoro di tesi, verrà utilizzata una blockchain *permissioned* (privata). Una blockchain permissioned gode innanzitutto delle stesse proprietà di una blockchain pubblica, ovvero:

- è una rete peer-to-peer decentralizzata dove ogni partecipante mantiene una replica di un libro mastro di transazioni firmate digitalmente;
- le repliche sono mantenute sincronizzate attraverso un protocollo di consenso;
- sono presenti garanzie di immutabilità anche in presenza di partecipanti maligni.

Quello che una blockchain permissioned aggiunge ad una blockchain pubblica è un meccanismo di controllo degli accessi integrato nei client, così che i peer possano essere aggiunti nella rete sulla base di un valore di controllo (un certificato o un indirizzo ad esempio). La scelta di utilizzare blockchain permissioned si adatta alla volontà di aziende, governance e pubbliche amministrazioni che vogliono utilizzare il livello tecnlogico e la sicurezza raggiunte dalle blockchain, mantenendo al tempo stesso la loro presenza all'interno della rete, necessaria per mantenere il controllo sui partecipanti. Queste blockchain sono sì adatte a gestire moneta, ma le loro potenzialità si manifestano soprattutto in altri campi come la condivisione di dati medicali (i quali non devono essere alterati in alcun modo) o la catena di distribuzione di farmaci (per evitare frodi a qualsiasi livello della supply-chain, dalla produzione alla vendita al dettaglio).
Le blockchain permissioned imitano il processo di sicurezza utilizzato dalle blockchain pubbliche come Bitcoin, ma essendo il numero e l'identità dei partecipanti conosciuto, non c'è il bisogno della proof-of-work o della proof-of-stake, mentre utilizzano la stessa crittografia e le stesse strutture dati come i Merkle trees[12] per assicurare che transazioni non valide non siano aggiunte alla blockchain. I meccanismi di consenso più popolari includono Raft[13] e Juno[14]. Questi protocolli di consenso funzionano sulla base di un modello leader-follower, in cui per ogni blocco viene selezionato un leader che crea il blocco e lo aggiunge alla blockchain, mentre errori ed anomalie vengono automaticamente corretti dal sistema.
Anche le blockchain permissioned godono della robustezza tipica della loro controparte pubblica dato che mantengono la proprietà di essere ridondanti e quindi estremamente fault-tolerant. Ogni nodo processa le transazioni, perciò nessun nodo risulta indispensabile. Per questo motivo, e per la natura peer-to-peer della rete, la replicazione che nei database viene realizzata tramite tecniche apposite, nelle blockchain è un processo naturale dato che i nodi si sincronizzano a vicenda. Infine ogni volta che si effettua una transazione rispetto ai database tradizionali, i sistemi basati su blockchain devono farsi carico di ulteriori oneri:

- verifica della firma: ogni transazione deve essere firmata con uno schema di crittografia a chiave pubblica;
- meccanismo di conenso: i nodi della rete devono raggiungere il consenso;
- ridondanza delle operazioni.

Questo ovviamente si traduce in prestazioni minori rispetto ai database tradizionali. Le blockchain permissioned quindi cercano di unire le capacità di prestazione e riservatezza (tipiche dei database tradizionali) con i meccanismi di fault-tolerance

e disintermediazione (per quanto possibile nelle blockchain private) tipiche della blockchain.

7 Limiti delle Blockchain

Per quanto valida la tecnologia della blockchain come quella presentata ed implementata da Bitcoin, essa presenta dei limiti nel caso in cui si voglia costruire applicazioni avanzate su di essa. Infatti Bitcoin presenta un linguaggio di scripting estremamente limitato, che soffre dei seguenti limiti:

1. *mancanza di completezza del linguaggio di Turing*: ovvero, mentre esiste un grande sottoinsieme di calcolo che il linguaggio di scripting del Bitcoin supporta, quest ultimo non supporta tutto. Infatti troviamo la mancanza dei «loop»;

2. *cecità del valore*: non c'è un modo per un UTXO di fornire un controllo a setaccio sull'ammontare ritirabile;

3. *mancanza di stato*: UTXO può essere sia speso che non speso, non c'è possibilità per scripts di mantenere altri stati interni oltre questo.

La soluzione sta nelle blockchain di seconda generazione che riprendono un concetto teorizzato negli anni '70, gli Smart Contract, e lo implementano all'interno della loro rete peer-to-peer.

Capitolo 4

Ethereum

1 Introduzione e Smart Contracts

Gli Smart Contract sono stati oggetto di sperimentazione già negli anni '90 quando le tecnologie hanno permesso di attuare forme di sperimentazione di Smart Contract, ma l'idea di Contratto Intelligente risale in realtà alla metà degli Anni '70. Il termine adottato all'epoca non era quello di Smart Contract, ma il concetto era sostanzialmente quello che ha portato ai contratti intelligenti. Infatti all'epoca l'esigenza atteneva alla necessità di gestire l'attivazione o disattivazione di una licenza software in funzione di alcune condizioni molto semplici. La licenza di determinati software venne di fatto gestita da una chiave digitale che permetteva il funzionamento del software se il cliente aveva pagato la licenza e ne cessava il funzionamento alla data di scadenza del contratto è stato il primo esempio di uno Smart Contract.
Ethereum[15] nasce invece nel 2013 per opera di Vitalik Buterin, uno sviluppatore di origini russe che ha unito la sua competenza di programmatore a quelle di ricercatore nell'ambito delle cryptocurrency. Lo scopo di Ethereum è quello di creare un protocollo alternativo per la costruzione di applicazioni decentralizzate in situazioni in cui sono essenziali: un tempo rapido di sviluppo, un alto gradi di sicurezza nelle applicazioni e la capacità di far interagire tra di loro in modo efficace applicazioni differenti. In definitiva, Ethereum è una blockchain con un linguaggio di programmazione costruito al suo interno, Turing completo, che permette a chiunque voglia di scrivere Smart Contracts e applicazioni decentralizzate (o DAPP) nelle quali è possibile stabilire:

- le proprie regole di proprietà;
- i formati delle transazioni;
- le funzioni di transizione di stato.

L'uso delle risorse computazionali di Ethereum è remunerato con una speciale "moneta virtuale" denominata *Ether* che rappresenta essa stessa sia la potenza elaborativa necessaria per produrre i contratti sia la cryptovaluta che permette di "pagare" per la realizzazione dei contratti[1]. Ether è fondamentalmente un token che viene trattato come cryptocurrency exchange con il ticker symbol di ETC.
Una generica applicazione Ethereum decentralizzata può essere schematizzata ad alto livello nel seguente modo:

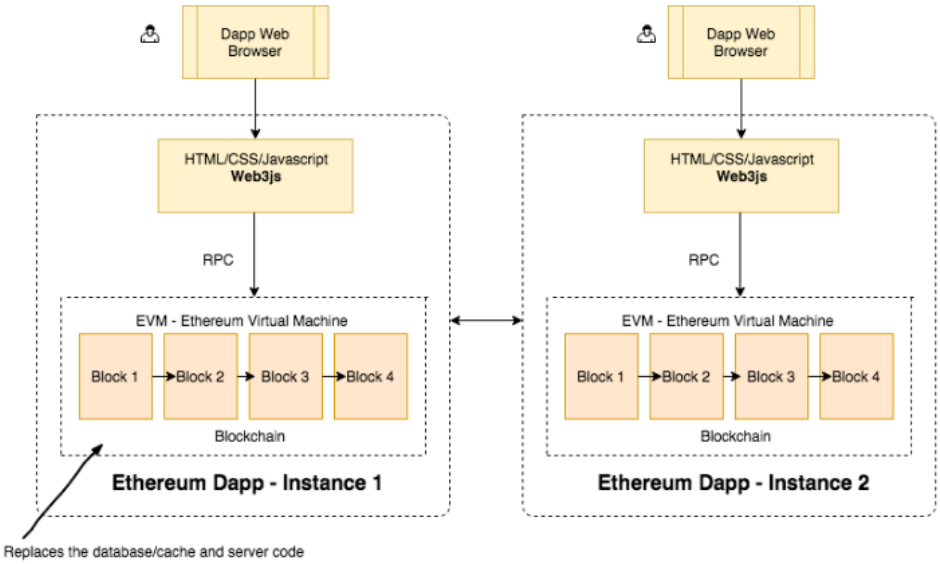

Figura 4.1: Esempio di applicazione decentralizzata

2 Ethereum Accounts

In Ethereum, lo stato è costituito da oggetti chiamati "accounts". Ognuno di essi ha un indirizzo di 20 byte e le transizioni di stato sono trasferimenti diretti del valore e dell'informazione tra gli accounts. Un account Ethereum, è composto dai seguenti campi:

1. il nonce: contatore utilizzato per assicurarsi che ogni transazione venga elaborata una sola volta;

2. il bilancio ether: rappresenta il conto corrente dell'account;

[1] in maniera simile a quanto avviene sulla blockchain di Bitcoin

3. il contract code: il codice del contratto che può essere eseguito dai nodi della rete. Contiene funzioni che possono essere richiamato, perciò può essere paragonato ad un oggetto di un linguaggio object-oriented;

4. lo storage dell'account.

In generale, esistono due tipi di accounts:

1. *accounts posseduti dall'esterno*: controllati da chiavi private. Questi non hanno codice ed è possono mandare messaggi esterni creando e firmando una transazione;

2. *accounts contratto*: controllati dal loro codice di contratto. Ogni volta che un account contratto riceve un messaggio (transazioni o messaggi) da altri contratti, il suo codice si attiva, permettendo a questo di leggere e scrivere verso uno storage interno e spedire altri messaggi oppure creare a sua volta contratti.
   Sono controllati da chiavi private.

Quindi tutte le azioni sulla blockchain Ethereum sono innescate da transazioni inviate da account controllati esternamente.

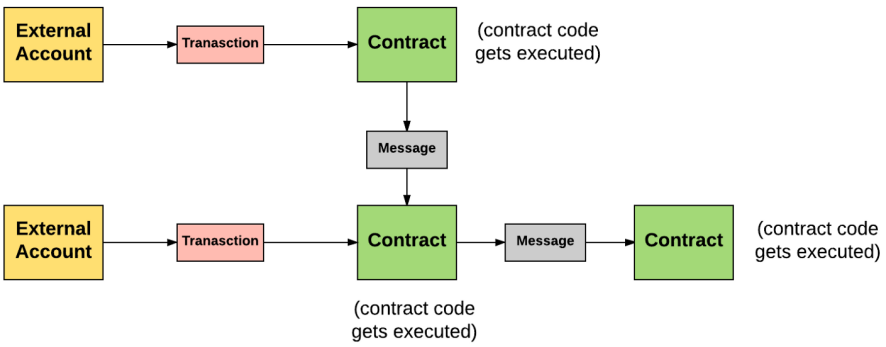

Figura 4.2: Esempio di interazione tra accounts

All'interno di Ethereum quindi gli account possono interagire tra di loro tramite messaggi oppure tramite transazioni.

3 Messaggi e Transazioni

3.1 Transazioni

Il termine *transazione*, in Ethereum, viene utilizzato per riferirsi all'insieme di dati firmati che contengono un messaggio da inviare da un account posseduto dall'esterno. Le transazioni contengono:

1. il destinatario del messaggio;
2. una firma che identifica il mittente;
3. l'ammontare di ether da trasferire al destinatario;
4. un campo dati opzionale;
5. un valore chiamato *STARTGAS*: rappresenta il massimo numero di steps computazionali che l'esecuzione della transazione può impiegare;
6. un valore chiamato *GASPRICE*: rappresenta la commissione che il mittente paga per lo step computazionale;
7. v, r, s: usati per generare la firma che identifica il mittente della transazione.

I campi (1.), (2.), (3.) sono campi standard previsti da qualsiasi criptovaluta. Il campo dati (4.) non ha una struttuda di default, potrebbe contenere un messaggio arbitrario oppure il codice per creare un contratto. Infatti la virtual machine di Ethereum ha un codice operativo con cui un contratto può accedere ai dati. Ad esempio, se un contratto sta funzionando come un servizio di gestione di ricette mediche basato su blockchain, potrebbe interpretare i campi che gli vengono passati come i campi della ricetta relativa in modo tale da poterli salvare correttamente nello storage.
I campi STARTGAS e GASPRICE sono essenziali per il modello di servizio anti-denial di Ethereum. Infatti al fine di prevenire loop infiniti, accidentali o malevoli che siano, o evitare spreco computazione di codice, ad ogni transazione è richiesto di impostare un limite per l'uso degli step computazionali di codice di esecuzione. L'unita fondamentale di computazione è il *gas*. Solitamente uno step computazionale costa 1 gas ma potrebbe anche costa di più in quanto dipende dal livello di complessità computazionale delle operazioni o dalla quantità di dati che deve essere conservata come parte dello stato.
Infine, anche in Ethereum è presente una commissione (di gas) per ogni byte della transazione. Questa commissione ha uno scopo ben preciso, ovvero quello di obbligare un qualsiasi aggressore della rete a pagare proporzionalmente per ogni

risorsa che essi vogliano consumare andando quindi ad includere la computazione, la larghezza di banda e lo storage.

3.2 Messaggi

I contratti hanno l'abilità di inviare "messaggi" ad altri contratti. I messaggi sono oggetti virtuali che non vengono mai serializzati ed esistono solo nell'ambiente di esecuzione di Ethereum (questi messaggi interni non sono quindi mai pubblicati sulla blockchain). Un messaggio è composto dai seguenti campi:

1. il mittente del messaggio (implicito);
2. il destinatario del messaggio;
3. l'ammontare di ether da inviare attraverso il messaggio;
4. un campo dati;
5. un valore STARTGAS.

Un messaggio quindi è simile ad una transazione, eccetto che esso viene prodotto da un contratto e non da un attore esterno. Un messaggio viene prodotto quando un contratto che sta eseguendo il codice effetta una *call*, che produce ed esegue appunto un messaggio. Come succede in una transazione, un messaggio comporta che l'account del destinatario esegua il proprio codice. In definitiva i contratti possono avere relazioni con altri contratti così come avviene con degli attori esterni e questo permette di avere un alto grado di libertà nella costruzione di applicazioni complesse.
In questo ambiente, la quantità di gas necessaria ad una transazione o ad un contratto, si applica alla totalità del gas consumato da quella transazione e da tutte le subesecuzioni. Ad esempio:

1. un attore esterno A trasmette una transazione a B con 1000 gas;
2. B consuma 600 gas prima di inviare il messaggio a C;
3. C consuma internamente 300 gas prima di ritornare;

Dopo questi step B può spendere solo altri 100 gas prima di rimanere a corto di gas.

4 Funzione Transazione di Stato

Ethereum, quindi, si presenta come una macchina a stati basata sulle transazioni. Infatti quando una transazione viene eseguita all'interno della blockchain, la transizione verso uno stato finale rappresenta, in qualsiasi momento, lo stato attuale di Ethereum. Ad esempio nella seguente figura possiamo osservare la transizione da uno stato "State" ad uno stato "State'":

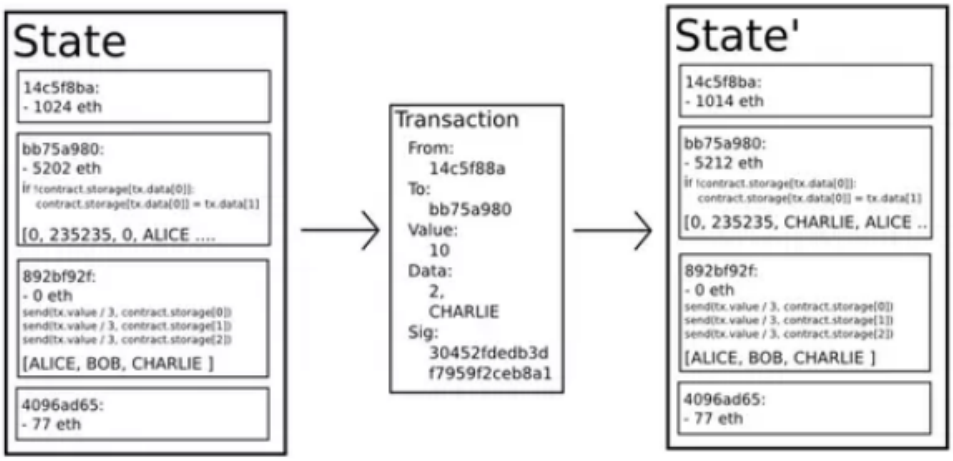

Figura 4.3: Esempio di transizione di stato

La funzione di transizione di stato di Ethereum può essere descritta, ad alto livello, nel seguente modo:

1. controlla se la transazione è ben impostata (possiede il giusto numero di valori), se la firma è valida e il nonce corrisponde a quello dell'account del mittente. In caso contrario si verifica un abort;

2. calcola la commissione di transazione come $STARTGAS * GASPRICE$ e determina l'indirizzo del mittente dalla firma. Sottrae poi la commissione dal bilancio dell'account del mittente e incremente il nonce del mittente. In caso in cui il bilancio da spendere sia insufficiente si verifica un abort;

3. inizializza $GAS = STARGAS$ e toglie una certa quantità di gas per byte per pagare i byte della transazione;

4. trasferisce il valore della transazione dall'account del mittente all'account del destinatario. Se l'account del destinatario non esiste ancora, lo crea. Se invece l'account destinatario è un contratto, esegue il codice del contratto per il completamento o fino all'esaurimento del gas;

5. se il trasferimento del valore fallisce perchè il mittente non ha abbastanza soldi, il codice dell'esecuzione esaurisce il gas e ripristina tutti i cambiamenti di stato tranne che per il pagamento della commisione e aggiunge le stesse commissioni sul conto del minatore;

6. se l'esecuzione termina correttamente, risarcisce al mittente le commissioni per il gas rimanente ed invia le commissioni pagate per il gas e consumante al miner.

4.1 Esecuzione del Codice

Il codice nei contratti Ethereum è scritto in un linguaggio di basso livello denominato *codice virtual machine Ethereum* oppure *codice EVM* che consiste in una serie di byte, dove ogni byte[2] rappresenta un'operazione eseguita all'interno dell'Ethereum Virtual Machine.
L'Ethereum Virtual Machine (EVM) è, quindi, l'ambiente di runtime per gli Smart Contract in Ethereum. L'ambiente risulta completamente sandboxato ed isolato il che significa che il codice in esecuzione al suo interno non ha accesso alla rete, al filesystem o ad altri processi. In generale, l'esecuzione del codice è un loop infinito che consiste nel realizzare ripetutamente l'operazione al contatore attuale del programma (che parte da 0) e incrementa il contatore di un'unità fino ad ottenere:

- la fine del codice;
- un errore o STOP;
- viene rilevata l'istruzione RETURN.

Le operazioni hanno accesso a tre tipi di spazio nel quale registrare i dati:

1. Stack: un contenitore di tipo Last-In-First-Out (LIFO) nel quale i valori possono essere spinti;
2. Memoria: un array di byte temporaneo (volatile) espandibile all'infinito;
3. Storage: memoria non volatile mantenuta come parte dello stato e caratterizzata dall'essere di tipo chiave-valore.

La struttura dell'EVM è la seguente:

[2]in realtà il codice viene solo successivamente compilato in bytecode. Lo sviluppatore può servirsi di linguaggi di alto livello come Solidity o Serpent

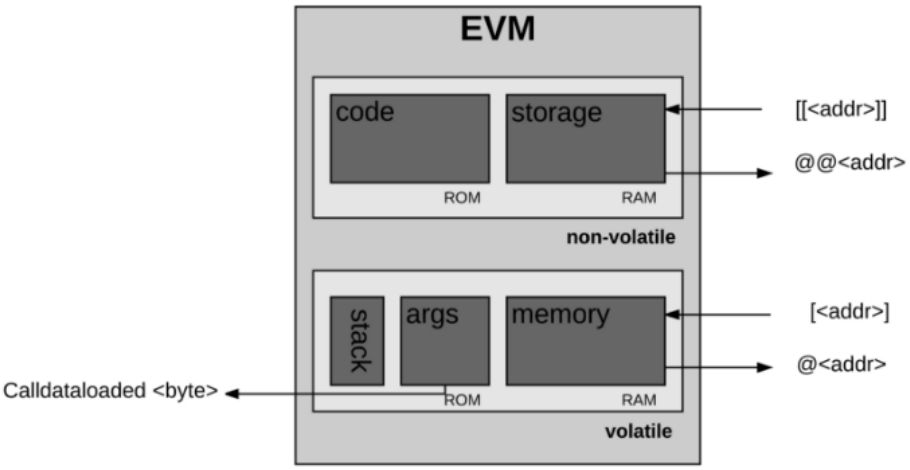

Figura 4.4: Schema dell'Ethereum Virtual Machine (EVM)

Il modello di esecuzione formale del codice EVM risulta estremamente semplice. Infatti mentre la virtual machine di Ethereum funziona, tutto il suo stato computazionale può essere definito dall'insieme di dati ($stato_del_blocco$, *transazione, messaggio, codice, memoria, stack, pc(i.e. program counter), gas*) dove $stato_del_blocco$ rappresenta lo stato globale che contiene tutti gli accounts e include i bilanci e lo storage.

5 Mining e Validazione

Una volta validate, tutte le transazioni devono essere inserite in un blocco che deve essere anch'esso validato all'interno della rete. Un blocco (block header) in Ethereum si presenta nel seguente modo:

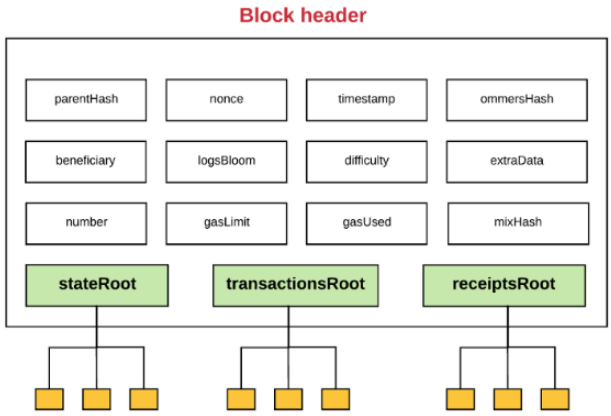

Figura 4.5: Blocco in Ethereum

Per una descrizione dettagliata della struttura del block header si rimanda all'Appendice A.
L'algoritmo di validazione del blocco in Ethereum funziona nel seguente modo:

- controlla se il precedente di riferimento essite ed è valido;
- controlla se la marca temporale del blocco è più grande di quella del blocco di riferimento precedente ed inferiore a 15 minuti nel futuro;
- controlla se il numero di blocco, la difficoltà del blocco, il transaction root, l'uncle root e il GASLIMIT sono validi;
- controlla se la proof-of-work sul blocco sia valida;
- sia $S[0]$ lo stato finale del blocco precedente;
- sia data la lista TX delle transazioni del blocco, con n transazioni. Per tutti i in $0..n-1$ sia $S[i+1]$ il risultato dell'applicazione della funzione transazione di stato sulla $TX[i]$ (ovvero ottengo il nuovo stato finale).
  Viene restituito un errore in caso di superamento del GASLIMIT o in caso di errore dell'applicazione;
- sia lo stato finale $S[n]$ aggiungendo però la ricompensa per il blocco pagata al miner;
- l'algoritmo controlla che il root state del merkle tree sia uguale allo state root finale fornito nell'intestazione del blocco. Se i due valori corrispondono il blocco viene dichiarato valido.

Come in Bitcoin, anche in Ethereum la fase del mining passa attraverso la proof-of-work chiamata "Ethash" (conosciuta precedentemente come Dagger-Hashimoto)[16]. L'algoritmo di valutazione, che a prima vista potrebbe sembrare inefficiente dato che registra l'intero stato di ogni blocco, risulta a livello di efficienza comparabile con quella di Bitcoin. Questo perchè lo stato viene registrato in una struttura ad albero e dopo ogni blocco solo una minima parte dell'albero necessita di essere cambiata. Quindi, in generale, tra due blocchi adiacenti la grande maggioranza dell'albero dovrà essere la stessa e quindi i dati possono essere registrati solo una volta e referenziati due volta tramite puntatori (ad es. gli hash degli alberi inferiori). Per raggiungere tutto questo Ethereum utilizza un tipo di albero conosciuto come *Patricia Tree*[3] , che consiste in una modifica ai Merkle Tree descritti in precedenza. Inoltre dato che tutta l'informazione sullo stato è una parte dell'ultimo blocco, non c'è bisogno di registrare tutta la storia della blockchain (ulteriore punto a favore rispetto alla blockchain di Bitcoin).

[3]Permette ai nodi di essere non solo modificati con efficienza, ma anche inseriti e cancellati

Capitolo 5

Software Ethereum-based per Blockchain private: Quorum

1 Introduzione

Come abbiamo scritto nei precedenti capitoli, gli Smart Contract si presentano come un libro mastro condiviso e replicato in ambiente Enterprise con la promessa di migliorare l'efficienza e diminuire i costi comparati con le altre soluzioni fornite da sistemi enterprise pre-esistenti[1] basati sulla logica di business duplicata e su un protocollo del consenso basato sulla riconciliazione. Però gli attuali sistemi basati su smart contracts non sono capaci di fornire la privacy di dati essendo lo stato delle transazioni e degli smart contract esposti in chiaro nella rete. Questo lavoro di tesi concentra la sua attenzione su una blockchain che è stata costruita sulle specifiche di Ethereum e poi modificata per ottenere i requisiti di privatezza in ambiente enterprise, dove avviene lo scambio dati sensibili (come ad esempio i dati medici).
Quorum[17] è un libro mastro distributo, privato, permissioned ed Ethereum-based che è stato sviluppato da J.P. Morgan per fornire all'industria dei servizi finanziari un'implementazione di Ethereum basata su permessi che possa supportare la privacy di contratti e transazioni. Quorum include quindi un fork minimo del client Ethereum (ovvero geth) facendo così leva sui traguardi tecnologici raggiunti dalla comunità degli sviluppatori di Ethereum.[2] Le feature principali di Quorum, che corrispondono anche alle principali aggiunte rispetto alla blockchain pubblica Ethereum, sono:

1. privatezza di contratti e transazioni. Quorum supporta transazioni e contratti

[1] Ad esempio Zcash
[2] La libreria go-ethereum e i suoi file binari sono licenziati sotto la GNU Lesser General Public License v3.0

privati attraverso la separazione tra stato pubblico e privato servendosi di *constellation* uno strumento per lo scambio di messaggi criptati peer-to-peer per lo scambio diretto privati tra i partecipanti della rete;

2. meccanismo di consenso basato sul voto multiplo. Essendo quorum una blockchain privata i partecipanti sono noti e non è strettamente necessario utilizzare la proof-of-work;

3. gestione dei permessi sia a livello di rete che a livello di singolo peer. Il permissioning viene realizzato utilizzando la logica degli smart contracts;

4. performance maggiori rispetto alla sua controparte pubblica.

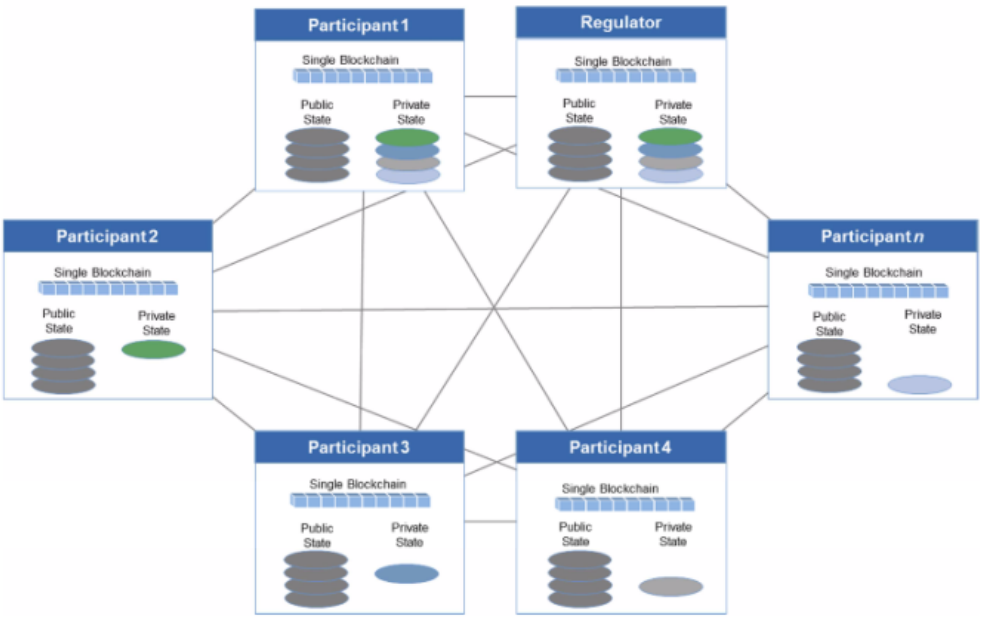

Figura 5.1: Diagramma di Quorum

Lo scopo principale degli sviluppatori è stato quello di riutilizzare per quanto possibile l'attuale tecnologia minimizzando i cambiamenti rispetto all'attuale versione di Ethereum sia per favorire i tempi di sviluppo e deploy delle applicazioni decentralizzate sia per minimizzare lo sforzo di mantenere sincronizzata la code-base comune di Ethereum presente in Quorum. Inoltre, anche se è stata pensata in prima istanza per implementare servizi finanziari in cui si vuole comunque mantenere una figura di tipo controllore all'interno della rete, è possibile utilizzare Quorum per ambiti esterni alla finanza in quanto mantiene tutte l'espressività a livello di codice della blockchain Ethereum.

2 Architettura

Attualmente Quorum include le seguenti componenti:

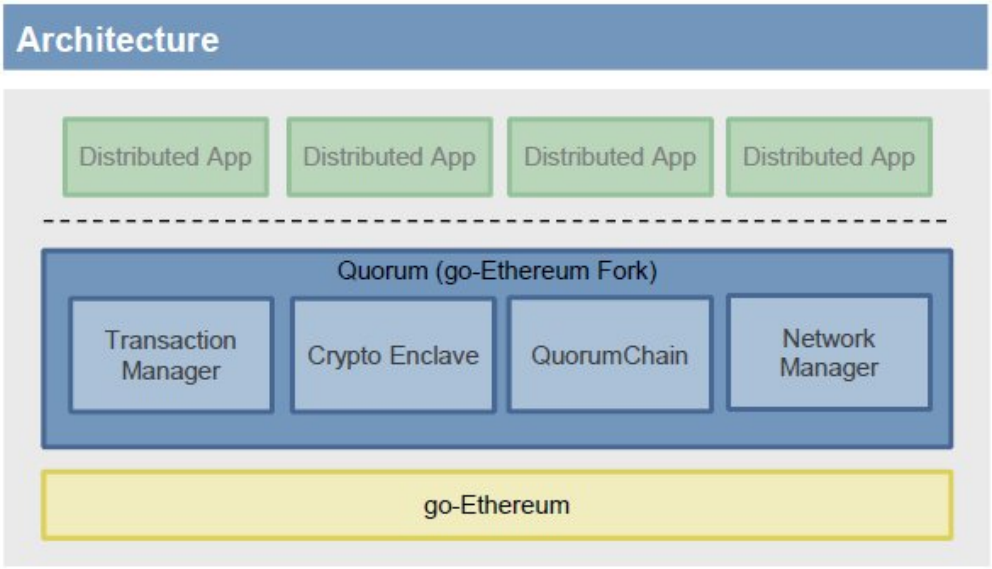

Figura 5.2: Componenti di Quorum

L'essenza di Quorum è quella di utilizzare la crittografia al fine di prevenire l'accesso ai dati sensibili a chiunque, eccetto per gli utenti coinvolti. Questa soluzione è stata realizzata utilizzando una singola blockchain condivisa e combinando l'architettura software degli smart contract (che fornisce la segmentazione dei dati privati) e le modifiche al client Ethereum (che include le modifiche al processo di proposta e validazione dei blocchi). Le sue componenti principali sono le seguenti:

Quorum Node Un nodo Quorum corrisponde al client geth modificato. Rispetto alla sua controparte pubblica contiene le seguenti modifiche:

1. l'algoritmo di consenso basato sulla proof-of-work è stato rimpiazzato da un meccanismo di tipo vote-based (più adatto alle blockchain private) denominato *QuorumChain*. Quorum attualmente offre anche delle alternative rispetto a QuorumChain stesso:
   - *Raft-based Consensus*: un modello di consenso per tempi di blocco maggiori, immutabilità delle transazioni e creazioni di blocchi on-demand.
   - *Istanbul BFT*: un algoritmo di consenso basato sul "Practical Byzantine Fault Tolerance" (PBFT).[18]
2. il layer P2P è stato modificato al fine di permettere connessioni solo da/verso nodi autorizzati;

3. sia la logica di generazione del blocco che la logica di validazione del blocco sono state modificate per sostituire il "Global state root" nel block header di Ethereum con il "Global public state root". Inoltre la logica di validazione del blocco è stata ulteriormente modificata per supportare le transazioni private;

4. il Patricia tree è stato diviso in: public state tree e private state tree;

5. la creazione delle transazioni è stata modificata per permettere che i dati della transazione vengano sostituiti con l'hash crittografico al fine di preservare la privatezza del dato quando richiesto;

6. Il prezzo del Gas è stato rimosso (anche se il concetto di gas rimane presente all'interno della blockchain).

Constellation Constellation è un sistema general purpose per trasmettere informazioni in modo sicuro. Può essere paragonato ad una rete di MTA (Message Transfer Agents) dove i messaggi sono criptati tramite il software "Pretty Good Privacy (PGP)". Non è un modulo pensato specificatamente per le blockchain ma è utilizzabile ogni volta che si vuole scambiare messaggi "sigillati" individualmente all'interno di una rete di partecipanti. Questo componente è composto da due sotto componenti: il nodo Constellation (che viene utilizzato nell'implementazione standard del *PrivateTransactionManager*) ed Enclave.

Il Transaction Manager. È responsabile per la privacy delle transazioni. Esso infatti conserva e permette l'accesso ai dati criptati delle transazioni, lo scambio di payload criptati con i transaction manager degli altri partecipanti e, al tempo stesso, non ha accesso ad alcuna chiave privata sensibile. Utilizza Enclave per le funzionalità crittografiche. Ogni Quorum node possiede il suo Transaction Manager che dialoga con le sue controparti distribuite sugli altri nodi.

Enclave. Abbiamo visto come i protocolli di ledger distribuiti facciano tipicamente uso di tecniche crittografiche per l'autenticità delle transazioni, l'autenticazione dei partecipanti e la conservazione storica dei dati (attraverso una catena di hash crittografici dei dati). Al fine di ottenere una separazione di compiti e al tempo stesso di garantire un miglioramento delle performanze attraverso la parallelizzazione delle operazioni che riguardano la crittografia, la maggior parte del lavoro crittografico inclusa la generazione di chiavi simmetriche e la cifratura/decifratura dei dati è delegata appunto ad Enclave.
Questo componente lavora quindi insieme al transaction manager per rafforzare la privacy attraverso la gestione della cifratura/decifratura in modo isolato. Esso

quindi si comporta a tutti gli effetti come un "Hardware Security Module" (HSM) isolato da tutti gli altri componenti.

Il Network Manager si occupa del controllo degli accessi alla rete, permettendo quindi la creazione di una rete permissioned.

3 Transazioni

Un'altra delle feature chiave di Quorum, come abbiamo detto, è la privatezza delle transazioni. Quorum infatti introduce la nozione di *transazione privata* contrapposta alla classica *transazione pubblica*. Questo però non vuol dire che gli sviluppatori di quorum hanno introdotto un nuovo tipo di transazione stravolgendo quindi il modello di Ethereum, bensì hanno esteso l'Ethereum Transaction Model nel seguente modo:

- viene aggiunto un parametro opzione *privateFor*[3] che, quando specificato, contraddistingue una transazione come privata e i nodi della blockchain la tratteranno di conseguenza;
- viene aggiunto un nuovo metodo al Transaction Type, denominato *isPrivate*, necessario per identificare le nuove transazioni private.

Transazioni pubbliche Le transazioni pubbliche sono quelle transazioni in cui il payload è visibile a tutti i partecipanti della rete Quorum (hanno la stessa visibilità delle transazioni in Ethereum). Questo tipo di transazione viene eseguito nel modo standard di Ethereum ovvero se una transazione pubblica viene inviata ad un account che possiede il codice di un contratto, ogni partecipante eseguirà lo stesso codice e il loro stateDb verrà aggiornato di conseguenza. Il valore V della firma della transazione viene settato dal nodo Quorum a $27o28$.

Transazioni private Le transazioni private sono quelle transazioni in cui il payload è visibile solamente ai partecipanti della rete le cui chiavi pubbliche sono specificate nel parametro privateFor della transazione. Quando il nodo di Quorum incontra una transazione con un parametro privateFor non nullo, setta il valore V della firma della transazione a $37o38$. Le transazioni private non sono eseguite nella modalità standard di Ethereum. Infatti prima che il nodo di Quorum del mittente propaghi la transazione al resto della rete, rimpiazza il payload originale della transazione con un hash del payload crittografato che riceve dal modulo Constellation. I partecipanti che fanno parte della transazione potranno rimpiazzare

[3] privateFor è un array di chiavi pubbliche

l'hash con il payload attuale attraverso la loro istanza di Constellation, chiamare l'EVM per l'esecuzione ed il loro stateDb sarà aggiornato di conseguenza. Invece i partecipanti che non fanno parte della transazione potranno leggere solamente l'hash e non eseguiranno la transazione (la transazione sarà a tutti gli effetti saltata da loro). Quindi questi due set di partecipanti termineranno con differenti stateDb e non potranno raggiungere il consenso perciò Quorum, al fine di supportare questa "biforcazione" dello stato del contratto immagazzina lo stato del contratto pubblico in un *Public State trie* sincronizzato globalmente mentre immagazzina lo stato dei contratti privati in un *Private state trie* non sincronizzato globalmente. Inoltre un contratto privato può essere creato solo attraverso una transazione privata.
Per una descrizione del flusso di una transazione privata all'interno di Quorum, si rimanda all'Appendice B.

4 Consenso in Quorum

Quorum[4] supporta il meccanismo di consenso denominato *QuorumChain*, un algoritmo a tempo a maggioranza di voto che utilizza:

1. uno smart contract per governare il consenso e gestire chi può partecipare al consenso;
2. transazioni Ethereum per propagare i voti attraverso la rete;
3. validazione della firma di Ethereum per validare le firme ricevute dai nodi *maker* e *voter*.

Quorum utilizza nel meccanismo di consenso una divisione di ruoli tra i nodi. Infatti all'interno della rete possono esistere nodi con il ruolo di *Voter* il quale permette loro di votare quale blocco dovrebbe essere la testa canonica ad una particolare altezza. Il blocco più recente con la maggioranza dei voti viene considerato la testa canonica di una catena ed un blocco viene considerato valido se e solo se ha ricevuto un numero minimo di voti dai Voters (supera la soglia minima).
La creazione dei blocchi invece è consentita solo ai nodi a cui è stato dato il ruolo di *Maker*. Un nodo con questo ruolo infatti può creare un blocco e firmarlo settando la sua firma nel campo *extraData* del blocco. All'atto dell'importazione del blocco come parte della procedura di Block Header Validation, i nodi verificano che il blocco sia stato firmato da uno dei nodi con il ruolo maker andando a cercare l'indirizzo di chi ha firmato il blocco nella lista dei makers validi contenuta nel *Voting contract*.
Un nodo può anche non ricevere un ruolo oppure riceverli entrambi.

[4]Nella sua versione studiata 1.5

4.1 Voting Smart Contract

Quorumchain è implementata nel contratto *Blockvoting* che è settato all'indirizzo $0x0000000000000000000000000000000000000020$ all'interno del Genesis Block[5]. All'interno del client Quorum sono hard-codati:

- quell'indirizzo;
- il bytecode pre-compilato per il contratto BlockVoting e il suo ABI[6] associato.

Nel caso in cui le regole del consenso all'interno del contratto BlockVoting dovessero essere aggiornate allora il nuovo codice dovrà essere compilato ed il client Quorum dovrà essere aggiornato per riflettere il nuovo codice. Tramite il contratto possono essere aggiunti o rimossi sia i nodi Voters che i nodi Makers e può essere configurato il numero minimo di voti (soglia) prima che un blocco venga selezionato come vincitore. Infine, come parte della validazione del blocco, il contratto deve determinare l'ultimo blocco che soddisfa il numero richiesto di voti (l'altezza canonica della catena) sul quale il blocco proposto si dovrà agganciare (il genitore del blocco proposto). Nel contratto sono inoltre registrati gli indirizzi dei nodi makers e voters. In particolare deve esserci almeno un nodo maker configurato nel contratto. In realtà sia l'insieme iniziale dei voti maker che dei voters è preconfigurato nel genesis block tramite il file di configurazione *genesis.json*, tuttavia una volta creata la rete i nodi maker possono aggiungere o rimuovere altri nodi maker inviando una transazione con la chiamata appropriata alla funzione contenuta nel contratto così come i nodi voters possono fare la stessa cosa verso altri nodi voters.

4.2 Observer node

Classe di nodi che non riceve ne il ruolo di maker ne quello di voter. Quindi non potranno prendere parte alla creazione di blocchi e alla votazione degli stessi ma bensì riceverà e validerà semplicemente i vari blocchi della rete.

4.3 Creazione e Votazione del blocco

Abbiamo visto come all'interno di una rete Quorum possano esistere più voti makers tuttavia, al fine di ridurre la probabilità per due o più maker di creare un blocco nello stesso momento, ogni maker genera un timeout che deve rispettare prima di poter creare un nuovo blocco. Il primo nodo maker ad uscire dal proprio timeout potrà creare il blocco e, una volta creato, genererà un nuovo timeout per se stesso e così via. Una volta che un nodo maker inizia il processo di creazione di un

[5] Il primo blocco della blockchain, contiene i parametri relativi all'inception della blockchain
[6] Application Binary Interface

blocco, gli altri nodi maker resetteranno il proprio timeout corrente, generandone quindi uno nuovo e aspetteranno la sua fine prima di provare a creare un nuovo blocco.
La durata del timeout cade casualmente tra un valore minimo e un valore massimo espressi in secondi che sono definiti nella *Blockmaker strategy* e può essere settato tramite la cli[7] flag *–minblocktime* e *–maxblocktime*. Saranno invece usati i default in caso in cui questi valori non saranno settati esplicitamente.
Dopo che un blocco è stato importato con successo come testa della catena, un nuovo blocco in attesa viene preparato sopra quel blocco. A questo punto tutte le transazioni processabili sono selezionate e applicate allo stato in sospeso nel nuovo blocco. Se il nodo è configurato con il ruolo Maker e gli viene ordinato di creare un blocco, prima convaliderà che l'hash del blocco genitore sia un hash canonico valido (ultimo blocco con numero di voti richiesto). Se questo blocco differisce dall'attuale testa locale, la creazione del blocco fallisce. Se invece il blocco è stato costruito sopra il blocco corretto, il nuovo blocco viene inserito nella catena di blocchi e propagato verso gli altri nodi. Durante questo processo, al fine di evitare il problema del *Chain halting* che può essere causato della limitata disponibilità dei nodi online, il nodo Maker invierà voti (se è configurato per essere anche un Voters) per il blocco principale al fine di soddisfare la soglia e consentire alla catena di progredire. Questo ha naturalmente implicazioni di controllo e dovrebbe essere gestito assicurandosi che i nodi dei voters soddisfino i loro obblighi verso la rete rimanendo disponibili.
La fase di voto invece deve avvenire entro un periodo preciso la cui durata è legata alla durata della creazione del blocco appena descritta. Dopo aver validato con successo un blocco, i voters trasmetteranno un voto per blocco chiamando il contratto BlockVoting. Nel caso in cui due nodi maker escano contemporaneamente dal loro timeout e creino entrambi un blocco, i nodi voters voteranno su entrambi i blocchi, ma quello con più voti alla fine della finestra di votazione sarà quello selezionato per la testa canonica della catena. Una descrizione schematica del processo di creazione, validazione e consenso in Quorum è contenuto nell'Appendice C. La struttura di un blocco Quorum includerà quindi:

1. *Global Transaction Hash*: l'hash di tutte le transazioni (pubbliche e private) presenti in un blocco;

2. *Public State root hash*: la controparte Quorum di quello che in Ethereum si chiama Global State root hash;

3. firma del blocco Maker nel campo *extraData*.

[7] Command Line Interface

5 Sicurezza in Quorum

Quorum è una blockchain di tipo permissioned ovvero è possibile controllare quale nodo può connettersi ad un dato nodo e anche quali nodi un dato nodo può chiamare. Al momento della scrittura di questo lavoro di tesi, Quorum gestisce i permessi a livello di nodo individuale tramite la cli flag *–permissioned* all'atto dell'avvio del nodo. Se questa flag viene settata il nodo cerca un file chiamato $< data - dir > /permissioned - nodes.json$ che contiene la whitelist dei nodi a cui questo nodo può connettersi e da cui può accettare connessioni. Quindi, con il flag abilitato solo i nodi che sono elencati nel file di configurazione json diventano parte di una rete, ma se il file dovesse essere vuoto a quel punto il nodo non potrà connettersi a nessun nodo ne accettare nessuna connessione in ingresso. Il file $permissioned - nodes.json$ ha una struttura precisa:

```
{
  [
    "enode://remoteky1@ip1:port1",
    "enode://remoteky1@ip2:port2",
    "enode://remoteky1@ip3:port3",
  ]
}
```

Ogni nodo avrà la sua versione di questo file.

5.1 Enclave

Enclave critta i payload che riceve dal Transaction Manager usando[8] *xsalsa20poly1305*[19] (payload container) e *curve25519xsalsa20poly1305*[19] (recipient box). Ogni crittografia del payload produce un payload container così come N Recipent box dove N è il numero dei destinatari specificato nel parametro "privateFor" della transazione. In particolare:

1. il payload container contiene il payload crittato con la chiave simmetrica (Cifratura simmetrica) e un nonce casuale;

2. il recipient box è la master key per il payload container crittato con la chiave pubblica di un destinatario utilizzando un nonce casuale (che corrisponde, in modo semplificato, al funzionamento di PGP ma usando le primitive crittografiche in NaCl);

In questa versione di Quorum la whitelist delle chiavi pubbliche viene definita manualmente e non viene eseguita la rotazione automatica delle chiavi. Questa funzionalità verrà inserita nelle successive versioni. Inoltre risulta anche mancante

[8] Sono primitive contenute nella "Networking and Cryptography library" (NACL)

un sistema di PKI essendo le chiavi generate casualmente e manualmente aggiunte alla whitelist (tutto questo viene effettuato e/o gestito da un operatore).
L'algoritmo per gestire le chiavi private è il seguente:

1. Data una password P;
2. Generato nonce casuale *Argon2i*;
3. Generato nonce casuale *NaCl secretbox*;
4. Eseguito lo stretch di P utilizzando *Argon2i* (e *Argon2i nonce*) in una *master key* (MK) di 32-byte;
5. Crittare la chiave privata nella secretbox usando *secretbox nonce* e *Argon2i-stretched MK*.

Capitolo 6

Strumenti di sviluppo

1 Introduzione

L'applicazione che si vuole implementare è costituita da tre tipologie di entità: le prime corrispondo agli account dei *medici* che hanno il compito di creare e rilasciare le ricette mediche bianche dematerializzate ai pazienti che le richiedono; le seconde sono i *farmacisti* che hanno il compito di accertarsi che la ricetta dematerializzata non sia stata manomessa per procedere all'erogazione dei farmaci contenuti nella ricetta stessa; le terze sono gli account che, in questa implementazione, avranno il ruolo di *amministratore* (admin) con la possibilità di associare un determinato ruolo (medico o farmacista) agli account esistenti.
Questo tipo di sistema può essere realizzato utilizzando come infrastruttura una blockchain privata: questa catena infatti contiene un insieme di transazioni attestanti ogni ricetta emessa dai medici e garantisce l'immutabilità del contenuto delle ricette stesse, permettendo i controlli di integrità sui contenuti inseriti. Inoltre la blockchain viene condivisa da tutte le parti coinvolte che, previo permesso, all'atto dell'ingresso nella rete possono scaricare localmente la copia più aggiornata e da quel momento in poi possono interagire con essa con nuovi inserimenti. Questa tecnologia infine rende immediatamente disponibili ogni inserimento di ricetta medica che viene trasmesso ad ogni nodo della rete.
Per una maggiore velocità nello sviluppo del prototipo applicativo, esso è stato pensato come un applicativo web. Generalmente, la struttura di una web application può essere schematizzata come nella seguente figura:

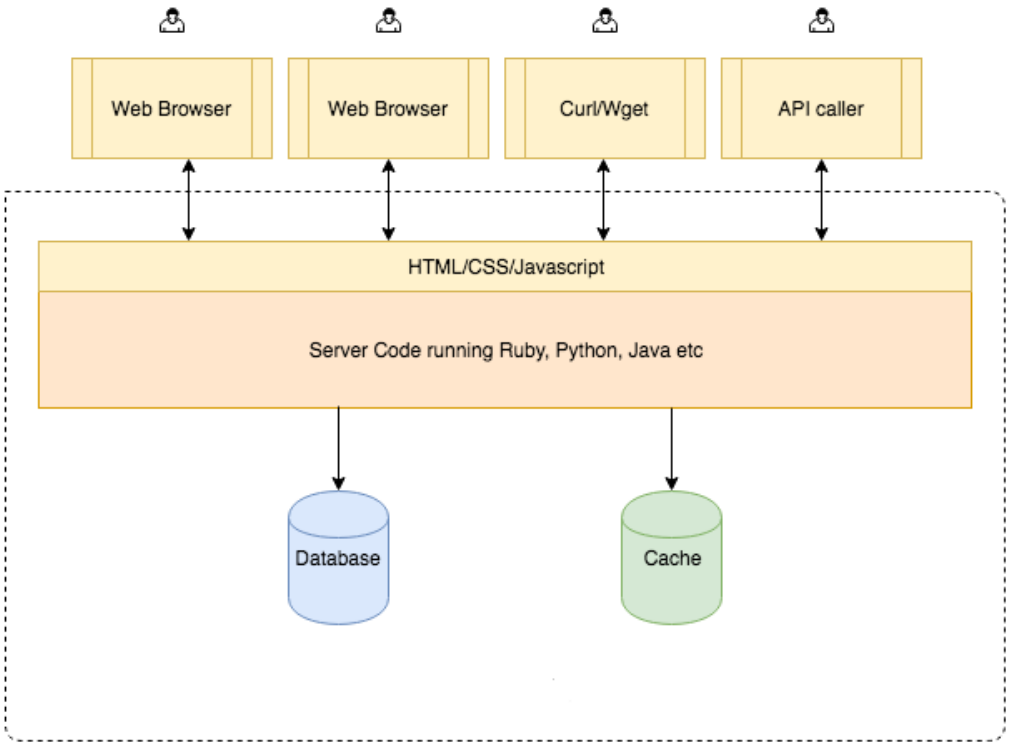

Figura 6.1: Web application generica

L'applicazione può essere ad esempio hostata su un servizio di hosting e tutti i client possono interagire con questa singola applicazione centrale. Un client può essere un browser oppure un'altra API che utilizza il servizio ecc. Quando il client effettua un richiesta al server quest ultimo prende in carico la richiesta, dialoga con il database e/o con la cache, legge/aggiorna/scrive il database[1] e serve il client. Questa tipologia di architettura è, al giorno d'oggi, una delle soluzioni tecniche più consolidate nell'ambito dello sviluppo del software. Però esistono anche determinate applicazioni dove sarebbe molto più utile se il database fosse ad esempio accessibile, in modo sicuro, da tutti senza dover dipendere da una terza parte. Questo è uno dei concetti chiave alla base di quelle che abbiamo definito precedentemente come *Applicazioni decentralizzate* (Decentralized Application o Dapp). La struttura di una generica applicazione decentralizzata può essere invece schematizzata come segue:

[1] CRUD Model. Create, Read, Update, Delete

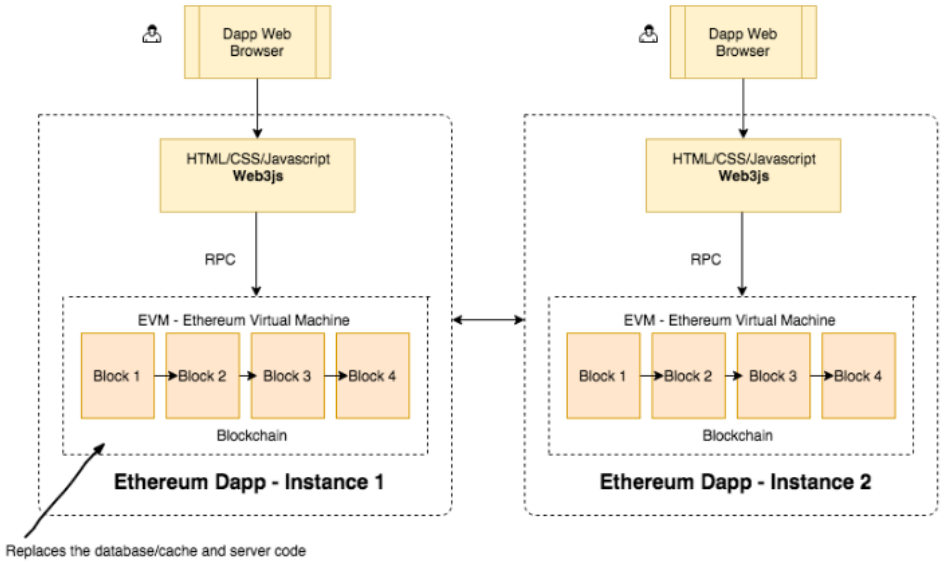

Figura 6.2: Decentralized application generica

Possiamo vedere come il client (nel nostro caso il browser) comunica con la propria istanza dell'applicazione. Non c'è alcun server centrale al quale i client si connettono. Ogni elemento in figura rappresenta una scelta tecnologica relativa allo sviluppo della dapp. In particolare:

1. i contratti sono scritti in *Solidity*, linguaggio per scrivere Smart contract Ethereum che girano sulla EVM;
2. *Truffle* è uno dei framework più utilizzati a disposizione dello sviluppatore per lo sviluppo e il testing in ambiente Ethereum;
3. *Web3.js* è una Api Javascript per i client Ethereum che permette di comunicare con il nodo locale attraverso Remote Procedure Calls (RPC);
4. la blockchain alla base della Dapp sarà Quorum.

2 Solidity

Solidità è un linguaggio di alto livello "contract-oriented" per l'implementazione di Smart Contract su varie piattaforme di Blockchain. È stato sviluppato da Gavin Wood ed è stato influenzato da C++, Python e JavaScript. Questo linguaggio viene compilato in bytecode eseguibile nell'Ethereum Virtual Machine (EVM) e supporta l'ereditarietà, le librerie e i tipi complessi definiti dall'utente.
Con Solidity, gli sviluppatori sono in grado di scrivere applicazioni che implementano

una logica di business autodefinita ed integrata nello Smart contract, fornendo una transazione non ripudiabile ed autorevole. Come specificato da Wood, il linguaggio è progettato attorno alla sintassi *ECMAScript* per renderlo familiare agli sviluppatori web ma, a differenza di ECMAScript, ha tipizzazione statica e tipi di ritorno variadici. Rispetto ad altri linguaggi attuali che vengono eseguiti su EVM come Serpent e Mutan, Solidity presenta delle differenze importanti:

- sono supportate variabili membro complesse per contratti, incluse mappature e strutture arbitrariamente gerarchiche;
- è supportata l'ereditarietà, inclusa l'ereditarietà multipla con la linearizzazione C3;
- introduce una ABI che facilita funzioni multiple con la type-safety[2];
- è supportato da un sistema di documentazione che specifica una descrizione, di tipo user-centric, delle ramificazioni di una chiamata del metodo nota come "Natural Language Specification".

Solidity inoltre ha a disposizione un ulteriore strumento molto potente per gli sviluppatori, chiamato *Remix*, che consiste in un IDE Browser-based con integrato il compilatore e l'ambiente di runtime di Solidity senza componenti server-side. Le sue caratteristiche principali sono:

1. sviluppo di smart contracts grazie all'editor integrato solidity;
2. debug dell'esecuzione di uno smart contract;
3. accesso allo stato e alle proprietà di uno smart contract già stato deployato;
4. debug di transazioni che hanno già effettuato il commit;
5. analisi del codice per minimizzare l'errore e per forzare le best practises.

[2]la "sicurezza di tipo" è il modo con cui un linguaggio di programmazione previene o avvisa gli errori di tipo

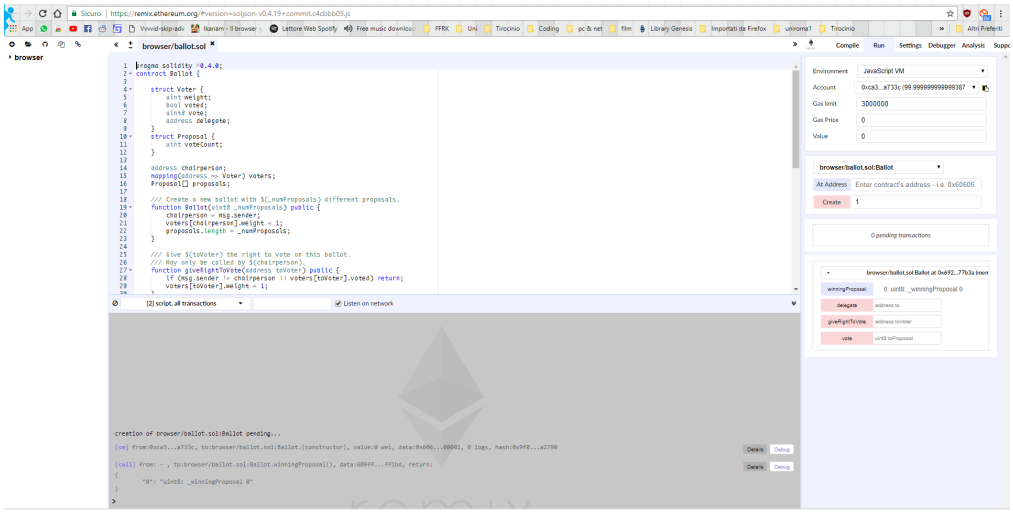

Figura 6.3: Remix IDE

3 Node.js & Npm

Node.js è un ambiente di run-time JavaScript, cross-platform e open-source per eseguire codice JavaScript lato server. Storicamente il linguaggio JavaScript è sempre stato principalmente usato come scripting lato client dove gli script erano integrati nelle pagine HTML per essere essere poi eseguiti dai motori JavaScript contenuti nei web browser degli utenti. Node.js permette al JavaScript di essere eseguito lato server per produrre pagine web dinamiche prima che esse vengano inviate al web browser dell'utente. Il modello di funzionamento di node è caratterizzato da un modello di I/O asincrono basato sugli eventi particolarmente adatto per le applicazioni web. La normale procedura di esecuzione di una applicazione server-side, scritta ad esempio in PHP, prevede lo scorrere di una serie di istruzioni che coinvolgono chiamate a servizi esterni come ad esempio query per un database, o l'accesso al filesystem per operazioni di lettura e scrittura. Tutti questi eventi sono bloccanti ovvero l'esecuzione dello script viene fermata finché non riceve una risposta dal componente esterno a cui l'ha richiesta. In tal modo il web server impiega la maggior parte del proprio tempo attendendo che i processi bloccati si completino, e questo porta ad una vistosa inefficienza nello sfruttamento delle risorse a disposizione. Tutto questo invece non succede in Node perchè, usando la programmazione asincrona, si elimina l'attesa e si eseguono le richieste successive, permettendo cosi un throughput e una scalabilità ottimizzate nelle web application.

Npm è invece un gestore di pacchetti per il linguaggio JavaScript. In particolare è il gestore di pacchetti JavaScript predefinito nell'ambiente Node.js. Esso è composto da un client a riga di comando e da un database online di package pubblici

(o moduli) denominato Npm registry. Un package è una directory che contiene uno o più file insieme ad un file chiamato package.json che contiene i dati relativi al pacchetto. Una web application che utilizza questa tecnologia per la gestione delle dipendenze in un progetto può contenere svariati pacchetti. Il vantaggio nell'usare un package manager risiede nella possibilità di aggiornare automaticamente tutte le dipendenze utilizzate evitando allo stesso tempo possibili "rotture" del codice causate da un aggiornamento manuale da parte dello sviluppatore.
Le tecnologie usate nello sviluppo dell'applicazione che si basano su Node.js sono:

- Truffle framework;
- Express.js.

3.1 Truffle Framework

Truffle, attualmente, è il framework di sviluppo più popolare per applicazioni Ethereum based[3]. Le feature che caratterizzano questo framework e che hanno portato alla sua scelta per lo sviluppo sono:

1. *Compilazione, linking, deployment e gestione dei binary degli smart contract integrata.* Il framework Truffle si occupa autonomamente della gestione degli artefatti dei contratti. Inoltre include il supporto per deploy custom, linking di librerie e applicazioni Ethereum complesse;
2. *Testing automatizzato dei contratti per un rapido sviluppo.* Truffle permette di scrivere test automatizzati sia in JavaScript che in Solidity per una maggiore rapidità di sviluppo;
3. Permette di scrivere *script di deploy custom* che riconoscono come l'applicazione possa cambiare nel tempo e che quindi permettono di avere uno sforzo di mantenimento minore;
4. *Gestione delle reti.* Truffle gestisce il deploy verso blockchain sia pubbliche che private. In questo modo lo sviluppatore non deve gestire manualmente gli artifatti di rete;
5. Per utilizzare Truffle è necessario avere Node.js installato e questo permette di scaricare facilmente migliaia di dipendenze per gli Smart contracts via NPM;
6. Truffle offre infine un ambiente ottimizzato nelle fasi di compilazione e test.

Lo scheletro di un progetto Truffle è composto da quattro cartelle principali:

[3]Truffle supporta allo stesso modo la Blockchian Quorum essendo un fork di Ethereum

- *contracts/*: cartella dove sono contenuti i contratti Solidity.
- *migrations/*: cartella dove sono contenuti gli scripts di migrazione.
- *test/*: cartella dove sono contenuti i file per il test dell'applicazione e dei contratti.
- *truffle.js*: file di configurazione di truffle.

3.2 Express.js

Express.js, o semplicemente Express, è un framework MVC per applicazioni web per Node.js, rilasciato come software libero ed open-source tramite MIT License ed è attualmente il framework server standard per Node.js. La filosofia di Express è quella di fornire un piccolo e robusto strumento per i server HTTP rendendolo la scelta preferita per le single page application, i siti web e le API. Come vedremo infatti, il compito principale di express all'interno del progetto sarà quello di gestire gli assett[4] dell'applicazione e definire le rotte che saranno utilizzate per le richieste che arrivano dal client. Una rotta può essere una richiesta di tipo GET (semplice richiesta di file,passaggio di pochi parametri) e di tipo POST(passaggio di molti parametri con attivazione di logica lato server) Nel nostro caso express ha in carico:

1. gestione dei file statici dell'applicazione;
2. la generazione del codice univoco NRE inserito nella ricetta dematerializzata ed utilizzato per la ricerca della ricetta stessa da parte dei farmacisti.
3. la generazione server side del pdf della ricetta dematerializzata.

La scelta di utilizzare di introdurre un elemento di tipo web server all'interno dell'architettura dell'applicazione è stata presa per motivi di sicurezza che saranno meglio spiegati durante la descrizione dell'implementazione dell'applicazione.

4 Web3.js

Web3.js è una collezione di librerie scritte in JavaScript che permettono allo sviluppatore di interagire con un nodo locale o remoto di Ethereum/Quorum utilizzando una connessione HTTP o IPC. Web3 è necessaria affinche la dapp lavori con la blockchain in quanto fornisce lo strumento di comunicazione verso il nodo locale attraverso Remote Procedure Calls (RPC). In particolare, la libreria contiene un oggetto specifico per interagire con le blockchain Ethereum-based, l'oggetto

[4] I file statici come le viste

web3.eth. Per utilizzare la libreria, una volta importata nel progetto, si deve creare un'istanza di web3 andando a settare il provider per comunicare con il nodo locale:

```
1  if (typeof web3 !== 'undefined') {
2    web3 = new Web3(web3.currentProvider);
3    } else {
4    // set the provider you want from Web3.providers
5    web3 = new Web3(new Web3.providers.HttpProvider("http://
       localhost:22000"));
6  }
```

A questo punto è possibile utilizzare le API dell'oggetto web3.

Capitolo 7

Configurazione dell'ambiente di sviluppo

1 Introduzione

Prima di procedere cn l'implementazione si è dovuti procedere a configurare l'intero ambiente di sviluppo locale relativo ai framework descritti nel capitolo precedente e al nodo locale della blockchain Quorum. Per lo sviluppo si è scelto di utilizzare un ambiente linux, precisamente Ubuntu 16.04 LTS.
La versione della blockchain Quorum utilizzata durante lo sviluppo di questa applicazione è stata la 1.5[1], caratterizzata da una procedura di configurazione divisa in più step che permette di avere un ambiente Quorum completamente funzionale composto da sette nodi indipendenti. Gli sviluppatori di Quorum consigliano, per questa versione, di utilizzare *Vagrant* e l'hypervisor di *Virtualbox*.

2 Vagrant

Vagrant è un software open-source progettato per cotruire e mantenere ambienti di sviluppo software portabili, capace di utilizzare i maggiori hypervisor attualmente utilizzati come VirtualBox (scelta predefinita all'atto dell'installazione), Hyper-V, Docker, VMware e KVM. L'idea alla base di questo software risiede nel fatto che mantenere l'ambiente di configurazione diventa particolarmente difficoltoso con l'aumentare delle tecnologie utilizzate in uno specifico progetto. Vagrant semplifica i passi della gestione delle configurazioni software al fine di aumentare la produttività dello sviluppo. Vagrant è scritto in Ruby ma il suo ecosistema supporta quasi

[1]Nel mese di ottobre Quorum è stata aggiornata ad una major version 1.8 per riallinearsi alla blockchain Ethereum (aggiornamento client geth in uso alla versione 1.7) e migliorare la propria gestione dei permessi e della privacy

tutti i maggiori linguaggi. Il software si basa sul concetto di "box". Una box può essere paragonata ad una scatola contenente il sistema operativo, e tutto quello di cui abbiamo bisogno come i software di base ed ogni loro configurazione associata. Invece le impostazioni del software che regolano la creazione di una determinata box sono racchiuse in un file scritto in ruby denominato "VagrantFile".
Quindi, una volta scaricato ed installato sia VirtualBox che Vagrant, il passo successivo è stato quello di customizzare i file di configurazione di Vagrant utilizzati nello sviluppo. Per completezza, i file sono stati inseriti nell' Appendice D relativa ai codici utilizzati nel progetto, nella sezione Vagrant, contiene:

1. la box utilizzata. Nel nostro caso *Ubuntu 16.04.3 LTS* (xenial);

2. la direttiva che specifica l'esecuzione di un file script di configurazione immediatamente dopo la creazione della box (bootstrap.sh);

3. il mapping delle porte TCP/UDP tra la macchina guest e la macchina host;

4. la quantità di memoria allocata alla box.

Il file realizzato per configurare la macchina dopo la sua creazione è denominato *bootstrap.sh* e contiene tutte le direttive per installare tutte le dipendenze di Quorum per avere una blockchain completamente funzionante. La sua esecuzione avrà come risultato:

1. l'aggiunta del repository ppa:ethereum/ethereum per avere l'accesso in download dei pacchetti Ethereum necessari in Quorum;

2. l'installazione di Constellation ed Enclave;

3. l'installazione di Golang, linguaggio di programmazione denominato "GO" scritto da Google. Il core di Quorum infatti è scritto in GO;

4. la build dei file sorgenti di Quorum scaricati dal repository di Quorum;

Il risultato di bootstrap.sh è un ambiente Quorum pronto per essere utilizzato. Una volta preparati i file, la procedura automatizzata di installazione e configurazione viene lanciata tramite il comando da shell: *vagrant up*. Terminata la configurazione, è possibile accedere alla macchina (e quindi alla blockchain) tramite il comando da shell: *vagrant ssh* che avrà il seguente output:

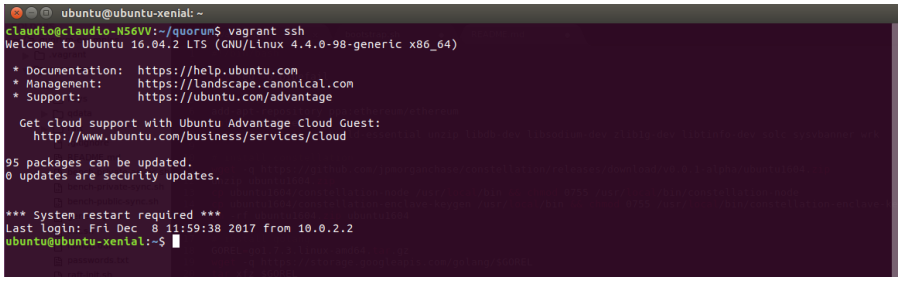

Figura 7.1: Terminale per interagire con la blockchain via ssh

3 Quorum

Una volta stabilita la connessione con la macchina virtuale contenente la blockchain, possiamo vedere come l'esecuzione del file bootstrap.sh descritto in precedenza, ha creato due cartelle nella home della macchina:

- una cartella *quorum* contenente tutti i file necessari al corretto funzionamento della blockchain;
- una cartella *quorum-examples* contenente i file per gestire e personalizzare una istanza locale di Quorum composta da sette nodi indipendenti.

Nella cartella "quorum-examples/7nodes" troviamo i file per interagire con la blockchain, come possiamo vedere nella seguente immagine:

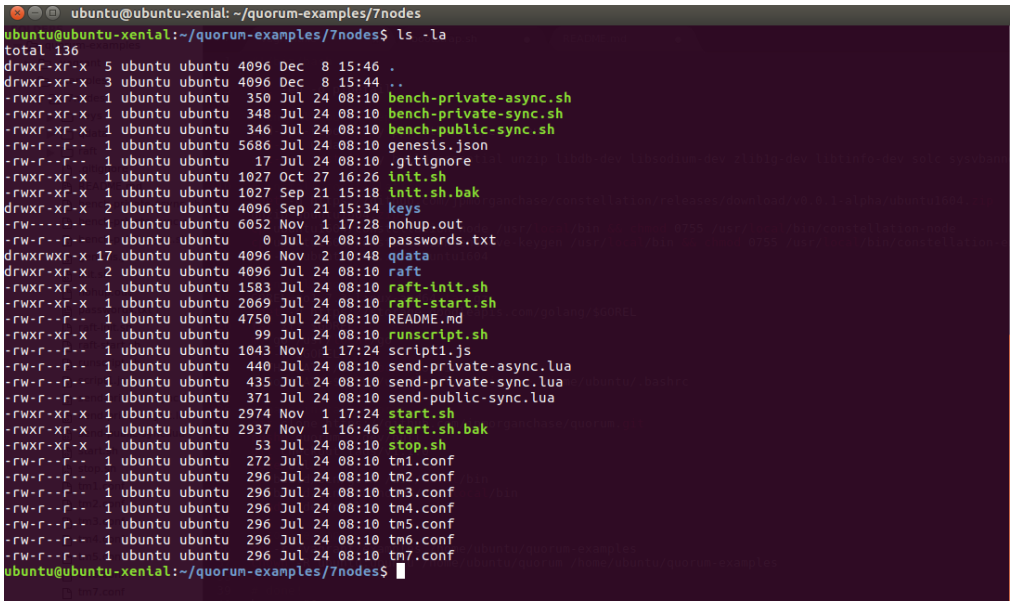

Figura 7.2: Struttura dei file della Blockchain Quorum

In particolare, i file che ci interessano in questa fase di avvio sono:

1. *init.sh.* Script da terminale per inizializzare la blockchain. Pulisce le cartelle con i file temporanei locali della blockchain ed inizializza i 7 nodi della blockchain;

2. *start.sh.* Avvia la blockchain (può accettare ulteriori parametri di configurazione a livello di singolo nodo;

3. *stop.sh.* Termina tutti i processi della blockchain in modo sicuro.

Questi file, in particolare init.sh e start.sh sono stati modificati per venire incontro alle esigenze dell'applicazione. Anche in questo caso, per l'intera struttura dei file, si rimanda all'Appendice D nella sezione Blockchain Quorum.

init.sh

Il file init.sh è stato riorganizzato al fine di avere, all'avvio della blockchain, due account correttamente configurati sul nodo 1 (dove sarà deployato successivamente il contratto). Per configurare correttamente gli account è necessario:

1. per ogni nodo corrispondente viene creata una cartella che andrà a contenere i *keystore* associati ad ogni account e si copiano le informazioni relative alle chiavi pubbliche degli account in ognuna delle cartelle, ove necessario;

2. si inizializzano i client geth (uno per nodo) tramite i file copiati, a seconda di come si vuole avviare la blockchain, e tramite l'utilizzo ulteriore del file *genesis.json.*

Nel caso invece si vogliano aggiungere ulteriori account, basterà generare tutte le informazioni e le chiavi per il nuovo account tramite Constellation e poi modificare il file init.sh (e lo start.sh di conseguenza).

start.sh

Il file start.sh contiente invece i comandi principali per lanciare correttamente l'esecuzione della blockchain. La sua esecuzione ha i seguenti risultati:

1. la configurazione del bootnode della blockchain con le direttive a tutte le componenti (eth, web3, quorum ecc);

2. l'avvio di un'instanza di Constellation per nodo;

3. l'avvio del bootnode;

4. l'avvio dei 7 nodi della blockchain. Per lo sviluppo è stata usata la seguente struttura di nodi:

- sul nodo 1 vengono configurati due account, uno dei quali con ruolo voter;
- sul nodo 2 vengono configurati due account, uno con ruolo voter ed uno con voto maker;
- nodo 3 configurato senza account;
- sul nodo 4 viene configurato un nodo voter;
- i nodi 5,6 e 7 vengono configurati senza account.

A questo punto la blockchain risulta correttamente configurata ed avviata. Gli account saranno poi associati agli attori dell'applicazione, come vedremo in seguito.

4 Node, Truffle & Express

Una volta configurata la blockchain è necessario configurare l'ambiente Node. Per farlo, si devono eseguire i seguenti comandi:

- ***sudo apt-get install nodejs***: verrà installata sulla macchina locale l'ultima versione di Node.js[2];
- ***sudo apt-get install npm***: verrà installato il package manager di Node.js.

A questo punto è possibile creare la cartella di progetto da dentro cui si andranno ad eseguire i seguenti comandi da terminale:

- ***npm init***. Questo comando andrà a creare il file *package.json* per l'applicazione, un descrittore json con le informazioni base del progetto e delle sue dipendenze. I campi inseriti sono:
  - *name*: nome dell'applicazione;
  - *version*: versione dell'applicazione che, insieme al name, crea un identificatore univoco.
  - *description*: descrizione dell'applicazione
  - *scripts*: dizionario chiave-valore che elenca gli script da lanciare (valore) associati agli eventi del ciclo di vita del progetto (chiave);
  - *dependencies*: contiene le dipendenze necessarie a runtime. Nel nostro caso contiene tutte le librerie per utilizzare correttamente le funzionalità offerte dall'applicazione.
- ***npm install -g truffle***: installa globalmente la libreria Truffle;

[2]La versione utilizzata nello sviluppo è stata la v.6.11.1

- ***npm install -g express***: installa globalmente la libreria Express.

A questo punto l'utente ha configurato tutto il necessario e può eseguire il comando ***truffle init*** che andrà ad inizializzare la cartella di progetto con lo scheletro di progetto di Truffle visto precedentemente. A questo punto, l'ultimo cambiamento da effettuare è la modifica del file truffle.js per farlo puntare verso la blockchain (nel nostro caso è stato scelto il nodo 1 precedentemente configurato), che sarà quindi il localhost sulla porta 22000 (già autorizzata a livello di permessi).

5 Deploy di un contratto sulla blockchain

Una volta configurato truffle per inviare i contratti sulla blockchain si dovrà effettuare quella che viene definita *migrazione*. La migrazione viene realizzata tramite file JavaScript che automatizzano l'attività di deploy del contratto sul nodo di Quorum. Questi file sono scritti in modo tale da poter essere facilmente mantenuti e modificati a seconda di come cambiano le necessità del progetto rispetto al deploy su una particolare blockchain. Si possono avere più file di migrazione all'interno del progetto.
Il comando per eseguire la migrazione è *truffle migrate*. Questo comando eseguirà tutti i file di migrazione inseriti nella cartella *migrations/*. Nel caso in cui tutte le migrazioni sono state eseguite con successo, l'esecuzione di truffle migrate partirà dall'ultimo file eseguito, andando ad eseguire solo le nuove migrazioni create. La migrazione verrà eseguita verso il nodo della blockchain specificato nel file *truffle.js*.
Per usufruire delle features delle migrazioni di Truffle è necessario avere all'interno del progetto il contratto di migrazione[3] e deve essere deployato come primo contratto nell'ordine di migrazione.
Il file di migrazione verso la blockchain Quoruma utilizzato nell'applicazione è il seguente:

```
1 var Prescriptions = artifacts.require("./Prescriptions.sol");
2 module.exports = function(deployer) {
3   deployer.deploy(Prescriptions, {privateFor: ["
      ROAZBWtSacxXQrOe3FGAqJDyJjFePR5ce4TSIzmJ0Bc="]});
4 };
```

Codice 7.1: Script di migrazione del contratto dell'applicazione

Possiamo vedere la presenza del parametro aggiuntivo, trattato nei capitoli precedenti, denominato *privateFor*. Questo parametro, specifico di Quorum, va ad indicare che il contratto è privato per tutti gli account specificati nell'array privateFor. Questo array andrà a contenere le chiavi pubbliche degli account della blockchain.

[3] Viene configurato in maniera autonoma dal comando *truffle init*

A questo punto, potranno interagire con il contratto privato tutti gli account la cui chiave pubblica è stata specificata all'atto del deploy e, inoltre, gli account che vorranno effettuare transazioni private dovranno specificare le chiavi pubbliche degli account con cui si vuole interagire all'atto dell'invio della transazione.
Lo script completo per il deploy e l'esecuzione dell'applicazione è il seguente:

```
truffle compile && truffle migrate --network quorum && NODE_ENV=
    development node server.js
```

Codice 7.2: Comandi per l'avvio dell'intero ambiente di sviluppo

Questo comando è composto da tre direttive:

- *truffle compile*: vengono compilati tutti i contratti presenti nella cartella *contracts/* o, eventualmente, solamente quelli modificati dopo l'ultima compilazione.
- *truffle migrate –network quorum*: vengono effettuate tutte le migrazioni dei contratti verso la blockchain in uso.
- *NODE_ENV=development node server.js*: server.js è il file principale per avviare l'istanza del web server Express (rappresenta il suo entry point)

Capitolo 8

Implementazione dell'applicazione

1 Obiettivo

L'obiettivo dell'attività di tesi è stato quello di sviluppare un prototipo di applicazione per la dematerializzazione delle ricette bianche, in grado di offrire:

1. meccanismi di garanzia di autenticità e validità delle ricette;
2. elevata affidabilità e tolleranza ai guasti del sistema;
3. accorgimenti specifici per il trattamento dei dati (sensibili).

Gli attori del sistema saranno:

1. I ***medici prescrittori*** che avranno il ruolo di erogare le ricette bianche dematerializzate e di rilasciare il relativo promemoria digitale (o cartaceo in caso di necessità) al paziente.
2. Le ***farmacie*** che avranno il ruolo di erogare la ricetta sulla base delle informazioni sul promemoria. In particolare le farmacie andranno a scansionare il qr-code contenuto sulla ricetta elettronica contenente una stringa rappresentante la ricetta stessa; prima dell'erogazione verrà utilizzata questa stringa per effettuare un controllo di integrità del contenuto della ricetta andando ad interrogare la blockchain per verificare se c'è stata una manomissione.
3. Una figura regolatrice nella veste di ***admin*** il cui unico compito nel prototipo sarà quello di assegnare il ruolo agli account della blockchain (medici prescrittori o farmacisti).

Le informazioni che saranno oggetto d'interazione per queste entità, saranno salvate in una blockchain permissioned che, nel caso in esame, è rappresentata da Quorum. In particolare, si propone la definizione di una rete che rispetti i seguenti requisiti:

- i nodi dei medici prescrittori possono essere rappresentati come full node (per via degli oneri come il mining).
- i nodi dei farmacisti possono essere rappresentati sia come full node che come lightweight node. La scelta dipende dall'adattabilità della soluzione implementativa e dall'eventuale riluttanza nell'allocazione dello spazio (seppur minimo) della blockchain.
- la figura dell'admin è stata inserita come gestore delle identità (associazione ruolo-account) all'interno della blockchain. Non andrà ad intervenire nel flusso dei dati tra medici prescrittori e farmacisti (nessun intermediario).
- nell'architettura di questa applicazione è stato introdotto, per motivi relativi alla sicurezza del dato, un application server con il ruolo di generatore del promemoria digitale.

Il risultato di questi requisiti si traduce nel seguente diagramma del sistema:

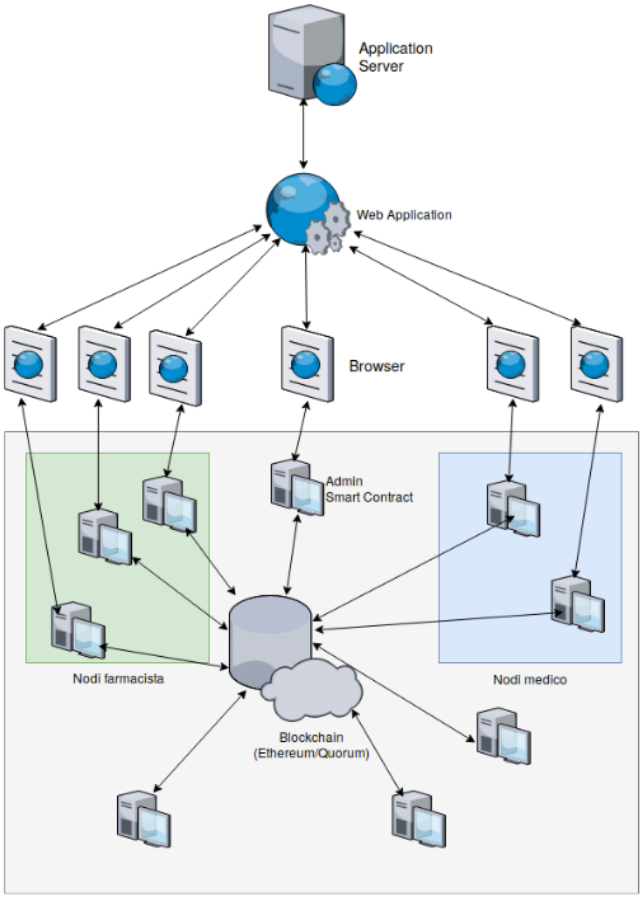

Figura 8.1: Diagramma del sistema

Per gestire la prescrizione e l'erogazione delle ricette dematerializzate nonchè l'assegnazione dei ruoli all'interno della rete, la blockchain viene personalizzata

mediante il deploy di uno specifico contratto di *erogazione*, contenente la logica dell'applicazione. Gli utenti potranno interagire direttamente con la blockchain attraverso il proprio browser. L'applicazione decentralizzata è stata pensata come una "single page application" con l'obiettivo di fornire una esperienza utente più fluida e simile alle applicazioni desktop dei sistemi operativi tradizionali. L'interfaccia è stata scritta utilizzando i linguaggi standard della programmazione web quali HTML5, CSS3 e JavaScript.

2 Ricetta medica

Il primo passo nell'implementazione dell'applicazione è stato quello di modellare l'oggetto "ricetta medica" secondo i requisiti enunciati. La ricetta medica dematerializzata si presenterà quindi in questo modo:

Figura 8.2: Struttura di una ricetta medica bianca dematerializzata

ed è composta dai seguenti campi:

1. *NRE*. Codice univoco identificativo della ricetta dematerializzata. Viene utilizzato per ottimizzare la ricerca di una particolare ricetta all'interno della struttura dove vengono salvate.

2. *Timestamp*: rappresenta il timestamp preso all'atto della creazione della ricetta.

3. *Medico*: rappresenta il nome e il cognome del medico prescrittore.

4. *Paziente*: rappresenta il nome e il cognome del paziente.

5. *Prescrizione* rappresenta il contenuto della prescrizione.

6. *Qr-code*. Si è scelto di utilizzare il Qr-code come tecnologia per velocizzare l'erogazione della ricetta elettronica da parte dei farmacisti. Il farmacista andrà ad effettuare la scansione del qr-code per avviare il processo di erogazione della ricetta.

7. *Account Medico*: rappresenta l'indirizzo dell' account del medico prescrittore.

Una volta ottenuta l'astrazione riguardante l'oggetto "ricetta medica", si è passati alla progettazione del contratto che regolasse la logica di funzionamento dell'applicazione sulla blockchain Quorum.

3 Il contratto prescrizione

La gestione della logica di funzionamento è stata realizzata mediante un contratto denominato *Prescription.sol* caratterizzato dalle seguenti proprietà:

1. una struttura per gestire i ruoli nel contratto (medico, farmacista e admin).

2. una struttura per gestire le informazioni di un account. Nell'applicazione saranno composte per semplicità da nome, cognome e ruolo.

3. una struttura per gestire le informazioni di una data ricetta. Nell'applicazione saranno composte da:

   - due variabili di tipo *address*[1] di cui una conterrà l'indirizzo sulla blockchain del medico prescrittore mentre l'altra onterrà l'indirizzo della farmacia che erogherà la farmacia. All'atto della creazione della ricetta questo dato particolare non è settato.
   - una variabile di tipo *bytes32* che conterrà il risultato dell'applicazione della primitiva crittografica *Keccak-256 SHA3* sul contenuto della ricetta generata dal medico.

4. per garantire invece che la modifica dello stato del contratto o l'invocazione delle funzioni del contratto vengano effettuate solo da determinati indirizzi, sono state inserite delle funzioni predefinite in Solidity denominate *modifiers*. Ogni ruolo definito nel contratto ha la propria funzione modifiers associata in modo tale da poter garantire il corretto funzionamento del contratto. I

[1] Contiene un valore di 20 byte che corrisponde alla grandezza indirizzo Ethereum

modifiers vengono usate per modificare il corpo di una funzione e in particolare vengono preposti all'atto della chiamata della funzione per cui sono specificati. Il controllo ha successo solo se l'indirizzo che ha chiamato la funzione è del tipo giusto.

5. il contratto infine contiene due strutture dati denominate *mapping*. In Solidity un mapping viene realizzato tramite la seguente sintassi $mapping(\_KeyType => \_ValueType)mapName$ dove $\_KeyType$ può essere la maggior parte dei tipo di dato[2] offerti in Solidity mentre $\_ValueType$ può essere qualsiasi tipo di dato. Questo tipo di dato può essere paragonato ad una tabella Hash. I due mapping sono stati definiti in accordo ai requisiti dell'applicazione in questo modo:

   - un mapping privato tra il codice univoco NRE e la struttura dati ricetta definita precedentemente. Questo mapping viene utilizzato per aumentare la velocità della ricerca della ricetta quando il farmacista procede ad effettuare il controllo su uno stato di una ricetta.
   - un mapping privato tra l'indirizzo di un account sulla blockchain e la struttura contenente che gestisce le informazioni di un account.

6. una serie di funzioni eseguibili che abilitano la logica del contratto. In particolare sono state implementate:

   - la funzione costruttrice del contratto che, all'atto del deploy sulla blockchain, setta l'account che effettua il deploy con il ruolo di amministratore. Si assume infatti che nella blockchain privata l'entità che procede al deploy del contratto sia l'entità di supervisione descritta nei capitoli precedenti.
   - le funzioni per recuperare i dati anagrafici di un account.
   - la funzione per recuperare il ruolo del mittente di una transazione.
   - la funzione che permette di creare l'associazione tra nome, cognome, ruolo e indirizzo. A questa funzione viene associato il modifiers *onlyAdmin*.
   - la funzione per inserire una ricetta nella blockchain. Prende l'hash del contenuto della ricetta, l'nre associato e crea l'associazione con l'indirizzo che ha inviato la transazione che corrisponde al medico prescrittore. A questa funzione viene associato il modifiers *onlyMedico*.
   - la funzione che recupera una ricetta a partire dal NRE. Questa restituisce l'indirizzo del medico associato alla ricetta, il suo nre e l'hash. Nel caso

[2] I tipi di dato non permessi come keyType sono mapping, array dinamici, struct ed enum

in cui la ricetta non esiste nella blockchain verrà restituito il valore $0x0000000000000000000000000000000000000000$.

- la funzione che permette l'erogazione della ricetta da parte del farmacista. A partire dal NRE della ricetta va ad aggiornare il secondo campo address della ricetta inserendo l'indirizzo dell'account che l'ha appena erogata (quello della farmacia).

4 Creazione di una ricetta medica elettronica

Abbiamo visto come, nella soluzione proposta, la ricetta dematerializzata viene concepita come un documento elettronico compilato dal medico prescrittore. Quest'ultimo infatti, tramite l'interfaccia specifica per il suo ruolo, andrà a riempire i campi della form corrispondenti alla ricetta elettronica. La pagina che viene presentata al medico è la seguente:

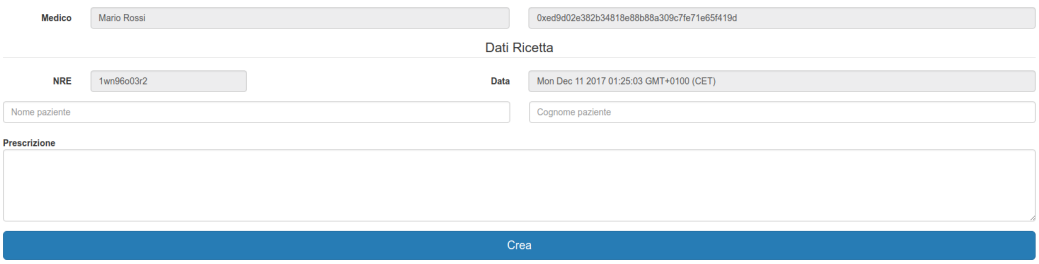

Figura 8.3: Schermata del medico prescrittore

Alcuni campi della form sono riempiti automaticamente e non possono essere modificati dal medico. In particolare le informazioni relative al nome e cognome del medico sono recuperate a partire dalla sessione di Web3.js attiva all'interno del browser (permette al browser di interagire direttamente con il contratto e la blockchain) mentre il timestamp viene generato lato client e il codice univoco NRE viene ricevuto dall'application server (tramite Express). La creazione di una ricetta elettronica da parte di un medico avviene nel modo seguente:

1. All'atto del caricamento della pagina del medico, l'application server provvede a generare ed inviare un nuovo NRE al client.

2. Il medico provvede a riempire i campi della form della ricetta elettronica.

3. Una volta riempiti tutti i campi il medico provvede a creare la ricetta cliccando sul bottone "crea". Il submit della form avvierà la seguente sottoprocedura (il dialogo con la blockchain è asincrono):

(a) viene parsato il campo data al fine di renderlo adatto alla trasmissione.

(b) tutti i campi della ricetta vengono serializzati in un oggetto json.

(c) l'oggetto json viene convertito in stringa per potervi applicare la primitiva crittografica sha3[3] messa a disposizione da Ethereum tramite le sue api.

(d) a questo punto viene effettuata una transazione in quanto si va a chiamare la funzione del contratto che si occupa dell'inserimento di una ricetta sulla blockchain.

(e) una volta ottenuto il riscontro effettivo dell'avvenuto inserimento della ricetta sulla blockchain viene effettuata una chiamata post verso l'application server al quale viene passato lo stesso oggetto serializzato inviato sulla blockchain.

(f) il server andrà prima a creare la struttura della ricetta elettronica (un documento in formato B6 landscape) inserendo tutti i campi come abbiamo visto nei paragrafi precedenti e, successivamente, andrà a creare il qr-code[4] contenente la stringa generata a partire dal json serializzato (così come è avvenuto nello step precedente) e lo inserirà all'interno della ricetta elettronica. Una volta generata la ricetta elettronica il server invierà il risultato al client permettendo al medico di salvare la ricetta elettronica oppure di creare una nuova ricetta. Si è scelto di dare la possibilità di stampa della ricetta dematerializzata (ripristinando così il promemoria essitente) per similitudine rispetto al sistema attuale tuttavia la scansione del qr-code può anche essere fatta tramite ricetta elettronica su smartphone permettendo così l'abbandono completo della carta nel processo di prescrizione delle ricette mediche.

Il procedimento appena descritto viene inoltre mostrato nel seguente diagramma delle sequenze:

[3]Ethereum vuole una stringa come argomento da passare a questa api

[4]Per la generazione del qr-code si è scelta una tra le tante librerire disponibili per Node. In questo caso *qr-image*ancona17

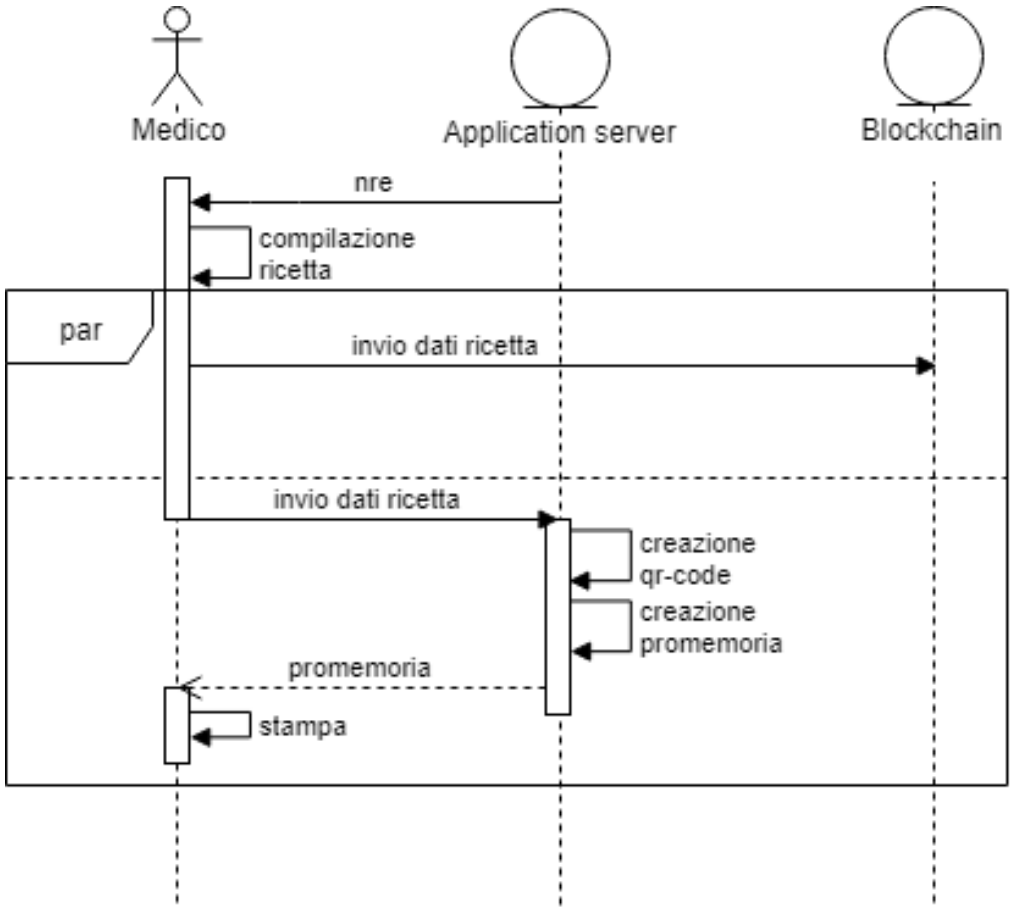

Figura 8.4: Creazione di una ricetta dematerializzata

5 Erogazione di una ricetta elettronica

Una volta che la ricetta viene correttamente salvata sulla blockchain e viene generato il corretto promemoria elettronico (ed eventualmente stampato) per il paziente, quest'ultimo può recarsi in una farmacia per richiedere l'erogazione della prescrizione che ha appena ricevuto. Come avviene per il medico prescrittore anche per il farmacista è stata un'interfaccia esclusiva per il suo ruolo così fatta:

Figura 8.5: Schermata del farmacista

L'interfaccia è stata divisa in due zone:

- Il box di sinistra andrà a contenere il feed della webcam. Sia per velocizzare la procedura di erogazione che per minimizzare l'errore dell'operatore (non è più necessario inserire i campi della ricetta manualmente per cercarla ed erogarla) è stato implementato uno scanner di codici qr-code che interagisce direttamente con la webcam del terminale dell'operatore.
  Lo scanner di qr-code è stato implementato completamente mediante HTML5 (HTML5 compliant) per renderlo il più possibile compatibile con i browser attuali senza il bisogno di installare tecnologie abilitanti di terze parti (e.g. Flash player di Adobe).

- la griglia di destra invece viene automaticamente popolata all'atto della scansione del qr-code andando quindi a contenere il riassunto, mostrato a schermo, del contenuto della ricetta estratto dal codice appena scansionato.

L'erogazione di una ricetta da parte del farmacista avviene nel seguente modo:

1. All'atto del caricamento della pagina del farmacista viene avviato lo scanner del qr-code e il suo feed sarà mostrato sull'interfaccia.

2. Nel momento in cui si va a scansionare un codice qr-code contenuto sulla ricetta elettronica il client provvederà a parsare il contenuto del qrcode per popolare la griglia e preparare i dati per la ricerca della ricetta sulla blockchain.

3. Una volta ottenuti tutti i campi della ricetta, il client ricrea la stringa che il client del medico prescrittore ha generato all'atto della creazione. Questo viene fatto perchè l'hash creato dal farmacista tramite i campi appena scansionati

sarà confrontato con l'hash salvato dalla blockchain dal medico. Per far questo il client chiamerà la funzione del contratto addetta alla ricerca di una determinata ricetta medica passandogli il codice univoco NRE. La ricerca può avere due esiti diversi:

- La ricetta non viene trovata. La funzione di ricerca restituisce valore $0x0000000000000000000000000000000000000000$ e il client notifica al farmacista che la ricetta che sta valutando non è valida in quanto non presente nella blockchain. Questo esito viene mostrato nella seguente figura:

Figura 8.6: Ricetta medica non valida

- la ricetta viene trovata all'interno della blockchain e il risultato della transazione è un oggetto che contiene l'account del medico prescrittore, l'hash della ricetta e il suo stato(se è stata o meno già erogata).

4. L'hash della ricetta creato dal farmacista sarà ora confrontato con l'hash della ricetta ricevuto dalla blockchain come prova di integrità e non manomissione del contenuto della ricetta. I possibili risultati di questo confronto sono:

- Ricetta valida. Il contenuto della ricetta corrisponde a quello salvato nella blockchain e non è già stata erogata. Il farmacista può procedere all'erogazione andando ad aggiornare lo stato della ricetta salvato sulla blockchain. Questa situazione viene mostrata nella seguente figura:

Figura 8.7: Ricetta medica valida

- La ricetta è valida ma è già stata erogata. Non è concessa la possibilità di erogare una seconda volta la ricetta elettronica. Questa situazione viene mostrata nella seguente figura:

Figura 8.8: Ricetta medica già erogata

- Ricetta medica non valida. Il contenuto della ricetta è stato alterato in qualche modo. L'erogazione viene rifiutata e la ricetta viene considerata non valida. Viene mostrata la stessa schermata di errore vista nel caso in cui la ricetta non viene trovata, ma con un codice di errore diverso.

Il procedimento appena descritto viene inoltre mostrato nel seguente diagramma delle sequenze:

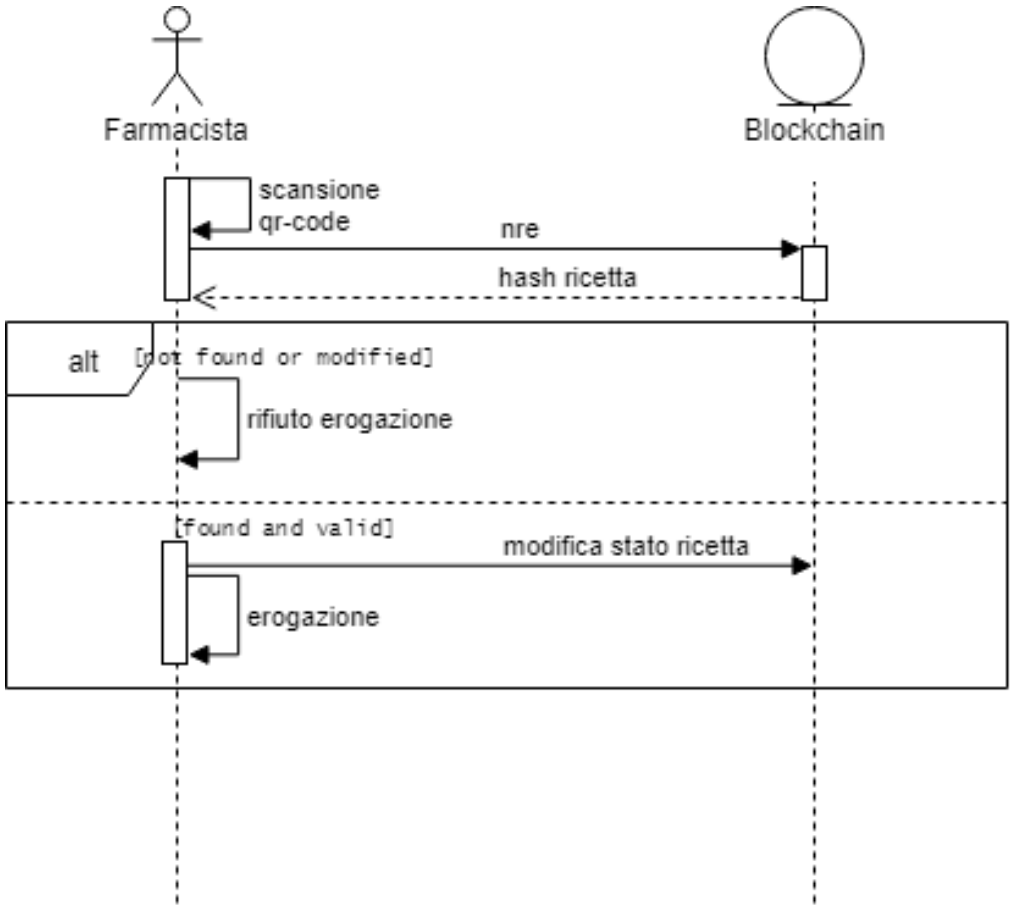

Figura 8.9: Creazione di una ricetta dematerializzata

6 Registrazione di nuovi account

L'ammministratore presente all'interno della blockchain ha l'unico compito di gestire le associazioni tra gli indirizzi degli account presenti sulla catena e il ruolo che essi andranno a ricoprire nell'applicazione. La sua figura è stata inserita per avere una gestione più rapida del mapping account-ruolo, considerando anche il fatto che una figura di supervisione, seppur senza possibilità di fare altre azioni, sia necessaria in quanto stiamo parlando di un ambito sanitario nazionale La sua schermata è la seguente:

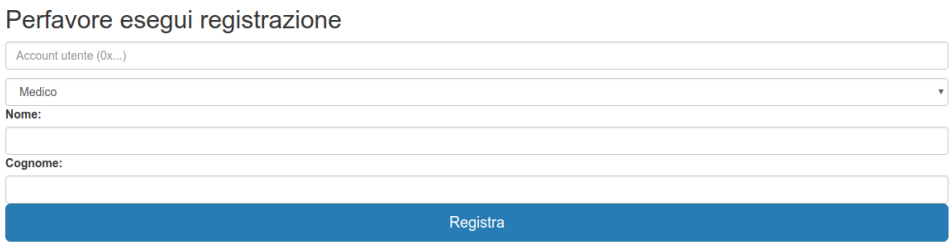

Figura 8.10: Assegnazione del ruolo da parte dell'amministratore

Capitolo 9

Considerazioni sulla sicurezza & conclusioni

1 Sicurezza

Quello realizzato nel corso dell'attività di tesi è un prototipo di un'applicazione per ricette mediche bianche dematerializzate che risponde ai *Requisiti centrati sulle informazioni*[1]ovvero quei requisiti informali che esprimono come dovrebbero essere gestite le informazioni all'interno di un sistema informatico. Questi requisiti sono:

1. *confidenzialità*: le informazioni devono essere accessibili solo a coloro che sono autorizzati. Questo è stato realizzato mediante l'utilizzo di una blockchain permissione privata che supporta sia transazioni pubbliche e private e non è interrogabile dall'esternon se non previa autorizzazione dell'autorità di supervisione. Inoltre il dato sensibile corrispondente al contenuto della ricetta elettronica non viaggia mai in chiaro sulla blockchain, ma quello che vedrebbe un eventuale utente malevolo sarebbe l'hash della ricetta.

2. *integrità*: le informazioni possono essere modificate solo da coloro che sono autorizzati. Questo requisito viene gestito a livello di logica di contratto tramite i modifier di accesso che vengono preposti alle transazioni quando si va ad interagire con l'inserimento della ricetta nella blockchain. L'applicazione supporta una divisione di ruoli che può essere eventualmente estesa per adattarsi alle nuove esigenze applicative.

3. *disponibilità*: le informazioni devono essere sempre accessibili in qualsiasi momento a coloro che ne hanno l'autorità o il permesso. Questa è una proprietà intrinseca della blockchain, data la sua natura decentralizzata.

[1]Modello CIA

Inoltre l'applicazione risponde ai *requisiti centrati sulle persone*[2] ovvero quei requisiti informali che esprimono i vincoli a cui dovrebbero essere soggetti gli individui che interagiscono con i sistemi informatici. Questi requisiti sono:

1. *autenticazione*: le persone devono provare la loro identità. Questo requisito viene supportato dalla tipica gestione degli account nella blockchain che sono identificati ed agiscono mediante la loro coppia di chiave pubblica-chiave privata. Ogni browser verrà configurato per utilizzare un solo account precedentemente autorizzato e le chiavi saranno conservate in modo sicuro. Al momento della scrittura del lavoro di tesi i modi possibili per raggiungere questo scopo sono:

   - l'utilizzo di un programma denominato "Metamask" compatibile con i maggiori browser in commercio, che permette di interagire con la blockchain dopo aver correttamente importato l'account mediante la chiave privata corrispondente (questo è possibile perchè in Metamask è capace di iniettare la sessione di web3.js all'interno del browser, permettendo così di effettuare transazioni da e verso la blockchain tramite l'account configurato in esso). Metamask è un programma di tipo "sandbox" ovvero una volta che è stata importata la chiave privata, se questa viene perduta risulta impossibile recuperarla tramite questo programma.
   - l'utilizzo di un browser modificato denominato "Mist". Attualmente si presenta come la principale alternativa ai browser in commercio. Mist ha già integrate tutte le librerie necessare per dialogare ed interagire con le blockchain e le applicazioni decentralizzate su di essa senza l'aggiunta di ulteriori programmi. A suo svantaggio va il fatto che deve essere configurato per interagire con le blockchain private (può risultare problematico) e rimane un programma con cui l'utente non ha familiarità.

2. *autorizzazione*. Le persone devono poter accedere ad una risorsa solo se autorizzate. Questo viene gestito ed implementato nella logica di contratto a livello di ruolo del singolo account.

3. *non ripudio*. Le persone devono essere responsabili delle proprie azioni e, in particolare, non possono negare quanto affermato precedentemente. Questo viene intrinsecamente supportato dalla blockchain in quanto ogni transazione viene firmata dal mittente (che sia essa pubblica o privata). Inoltre nel contratto sono state inseriti delle strutture dati come ulteriore traccia delle associazioni tra ruolo-account e account-ricetta.

[2] Modello AAA

1.1 Application server

All'interno dell'architettura decentralizzata è stata valutato ed implementato un elemento application server, un elemento tipico delle architetture client-server. Questa scelta che in prima istanza sembra vada ad inficiare l'implementazione decentralizzata è risultato necessario per questioni di sicurezza. Infatti il codice lato client, nonostante tutte le tecniche per pervenire XSS[3] e di offuscamento del codice, risulta sempre vulnerabile all'injection tramite la console del browser. Allora la logica di generazione della ricetta elettronica è stata spostata server-side (Express) al fine di evitare la modifica della logica di generazione di ricetta dematerializzata da parte di un utente malevolo e preservare così l'integrità dei file generati dal medico. Lo svantaggio di questa entità è che si comporta a tutti gli effetti come un *Single point of failure* ed è necessario prendere provvedimenti al livello di:

- gestione del carico del server tramite l'inserimento di load balancer (evitare i colli di bottiglia).
- gestione della disponibilità del server in caso di Attacco DoS[4] tramite l'applicazione di tecniche di tolleranza ai guasti (tecniche di replicazione e duplicazione).

2 Conclusioni

La scelta di una blockchain privata come Quorum, anche se complessa dal punto di vista della configurazione, risulta estremamente versatile nella gestione di sistemi complessi decentralizzati in cui sono necessarie determinati livelli di sicurezza. Inoltre si presenta anche come una valida soluzione per sviluppare software blockchain-based la cui logica risieda all'interno di contratti che vengono eseguiti sulla stessa blockchain dato che contiene dentro di se tutte le feature di Ethereum.
Inoltre l'utilizzo della tecnologia della blockchain si presenta come una valida alternativa all'utilizzo dei database tradizionali perchè sotto determinati requisiti riesce ad offrire maggiori garanzie rispetto ad essi. Infatti, anche se presente un'entità supervisionante, non sono presenti intermediari durante l'interazione tra gli attori principali del sistema (medici e farmacisti) e chiunque possiede l'autorizzazione (sia di ruolo che di peer) può interagire con le informazioni sulla blockchain mentre questo non è possibile nei database tradizionali dove solo l'autorità centrale può accedere ai nostri dati.

[3] Cross-Site-Scripting
[4] Denial of service

Inoltre, mentre nei database tradizioni risulta necessario applicare attivamente tecniche di replicazione e di tolleranza ai guasti, nella blockchain questo è intrinseco nell'architettura stessa. Perciò la scelta riguardante quale tecnologia da utilizzare deve essere presa in relazione agli obiettivi che si vogliono raggiungere.

2.1 Soluzione offerta

Nel merito dell'applicazione implementata per la gestione di ricette elettroniche dematerializzate sono stati raggiunti i seguenti obiettivi principali:

- ***performance***: rispetto alla controparte pubblica una blockchan privata offre performance maggiori in quanto utilizza meccanismi di consenso più adatti ad un ambiente permissioned e meno pesanti computazionalmente come la POW.
- ***disponibilità***: essendo una struttura di tipo peer-to-peer che cresce con l'aumentare degli utenti che partecipano all'attività al suo interno la possibilità di avere informazioni sempre disponibili aumenta.
- ***aggiornamento***: la blockchain risulta sempre aggiornata perchè le transazioni vengono sempre propagate attraverso i nodi, siano esse pubbliche o private.
- ***affidabilità***: il corretto funzionamento delle attività di mining viene supervisionato da un entità garante soprattutto nel caso di informazioni sensibili come i dati medico-sanitari.
- ***resilienza ai guasti***: la blockchain è intrinsecamente resiliente ai guasti.

Appendice A

Block Header Ethereum

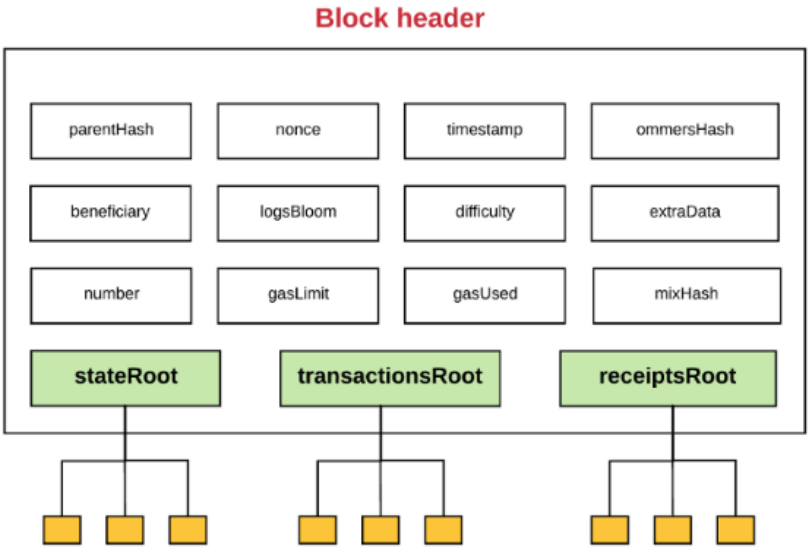

Figura A.1: Schema Blocco in Ethereum

In ethereum un blocco è composto da:

- il block header;
- le informazioni che riguardano l'insieme delle transazioni incluse nel blocco;
- un insieme di altri block header per l'attuale block ommers. Con "ommer" o "uncle block" si intende un figlio di un antenato che non è un antenato. Se A è uncle di B, B è nipote di A. Questi sono necessari per aiutare a ricompensare i miners quando sono trovate soluzioni duplicate di blocchi a causa dei tempi di blocco più corti di Ethereum (rispetto a Bitcoin ad esempio). Un uncle è una ricompensa più piccola rispetto ad un full block (se sono inviati oltre il blocco successivo, la ricompensa decresce rapidamente fino a zero dopo sette blocchi).

La parte che contiene la maggior parte delle informazioni per il corretto funzionamento della blockchain è il block header. Esso è composto da:

1. parentHash: l'Hash crittografico del block header genitore (questo è quello che rende l'insieme di blocchi una catena);
2. ommersHash: l'hash della corrente lista di block ommers;
3. beneficiary: l'indirizzo dell'account che riceve la ricompensa per minare il blocco;
4. stateRoot: struttura dati di tipo Tree. Contiene l'hash del root node dello state tree (così è facile per i light client verificare gli stati);
5. transactionsRoot: struttura dati di tipo Tree. Contiene l'hash del root node del tree che contiene tutte le transazioni listate nel blocco;
6. receiptsRoot: struttura dati di tipo Tree. Contiene l'hash del root node del tree che contiene le ricevute di tutte le transazioni listate nel blocco;
7. logsBloom: una struttura dati che contiene informazioni di log;
8. difficulty: indica il livello di difficoltà del blocco;
9. number: il numero del blocco corrente (il genesis block ha numero di blocco 0);
10. gasLimit: l'attuale gasLimit per blocco;
11. gasUsed: la somma del gas totale usato dalle transazioni in questo blocco;
12. timestamp: iltimestamp unix dell'inception di questo blocco;
13. extraData: dati extra relativi a questo blocco;
14. mixHash: un hash che, quando combinato con il nonce, dimostra che questo blocco ha portato avanti sufficiente computazione;
15. nonce: un hash che, quando combinato con il mixHash, dimostra che questo blocco ha portato avanti sufficiente computazione.

Appendice B

Schema transazioni Quorum

La figura mostra come viene processata una transazione privata in Quorum:

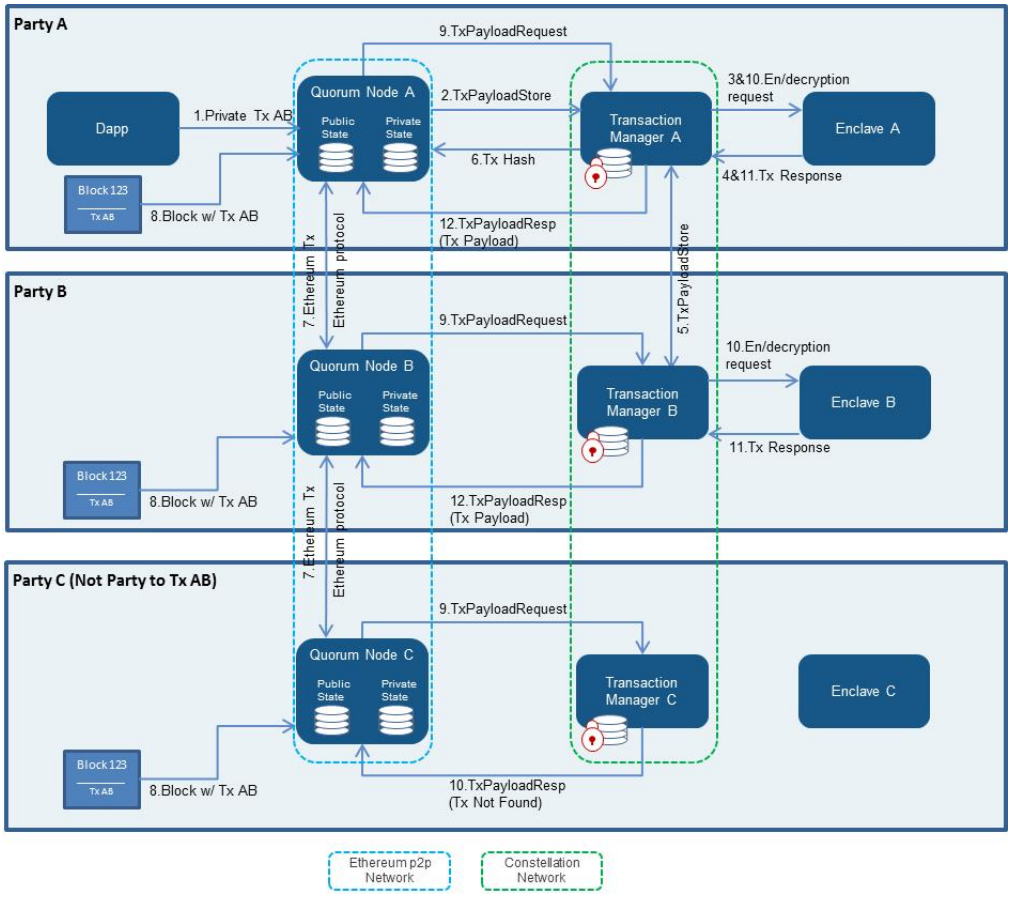

Figura B.1: Schema di una transazione privata Quorum

1. il party A invia una transazione al proprio nodo Quorum, specificando il payload della transazione e settando il "privateFor" con la chiave pubblica per i gruppi A e B.

2. il nodo Quorum del party A passa la transazione al relativo gestore delle transazioni, richiedendogli di memorizzare il payload.

3. il transaction manager del party A chiama il proprio Enclave associato per validare il mittente e crittare il payload.

4. l'Enclave del party A controlla la chiave privata[1] del party A e, una volta validata, effettua la conversione della transazione. Questo comporta:

   (a) generazione di chiave simmetrica e random nonce.

   (b) crittografia del payload della transazione e del nonce con la chiave simmetrica dello step precedente.

   (c) calcolo SHA3-512 del payload del passo precedente.

   (d) iterazione attraverso la lista dei destinatari della transazione, in questo caso i party A e B, crittando la chiave simmetrica dello step precedente con la chiave pubblica del destinatario (PGP).

   (e) viene ritornato il payload crittografato allo step 4.2, l'hash dello step 4.3 e la chiave crittografata (per ogni destinatario) dello step precedente al transaction manager.

5. il transaction manager del party A immagazzina il payload (criptato con la chiave simmetrica) e la chiave simmetrica criptata utilizzando l'hash come indice e poi trasferendo in modo sicuro: hash, payload criptato e la chiave simmetrica criptata con la chiave pubblica del party B al transaction manager del party B. Quest'ultimo risponderà con una risposta di tipo ack/nack. Se il party A non riceve risposta o riceve nack dal party B allora la transazione non sarà propagata nella rete.

6. una volta che la trasmissione dei dati al transaction manager del party B ha avuto successo il transaction manager del party A ritorna l'hash al nodo Quorum, il quale hash va a rimpiazzare il payload originale della transazione e cambia il valore V della transazione in 37 o 38. Questo valore indicherà agli altri nodi che questo hash rappresenta una transazione privata con un payload associato criptato.

7. a questo punto la transazione è propagata al resto della rete utilizzando il protocollo P2P standard di Ethereum.

8. un blocco contenente la transazione AB è creato e distribuito ad ogni party sulla rete.

[1] Cifratura asimmetrica

9. Elaborando il blocco tutti i party proveranno a processare la transazione. Ogni nodo Quorum riconoscerà un valore V di 37 o 38, identificandola quindi come transazione il cui payload deve essere decrittato, ed effettuerà una chiamata al proprio transaction manager per determinare se possiedono la transazione (utilizzando l'hash come indice per cercare).

10. dato che il party C non possiede la transazione riceverà un messaggio "NotRecipient" e salterà la transazione non aggiornando il suo stateDb privato. Invece la ricerca da parte dei party A e B avrà esito positivo identificando quindi il fatto che posseggono la transazione e procederanno a chiamare il proprio Enclave passando il payload criptato, la chiave simmetrica criptata e la firma.

11. Enclave valida la firma e poi decritta la chiave simmetrica utilizzando la chiave privata del party (posseduta da Enclave) , decritta il payload della transazione utilizzando la chiave simmetrica appena ottenuta e ritorna il payload decrittato al transaction manager.

12. il transaction manager dei party A e B allora invia il payload decrittato all'EVM per l'esecuzione del codice del contratto. Questa esecuzione aggiornerà solo il private stateDB del nodo Quorum. Una volta che il codice è stato eseguito, viene eliminato per far sì che non sia disponibile a meno di rieseguire il processo di decrittazione

Appendice C

Conseus Process Flow

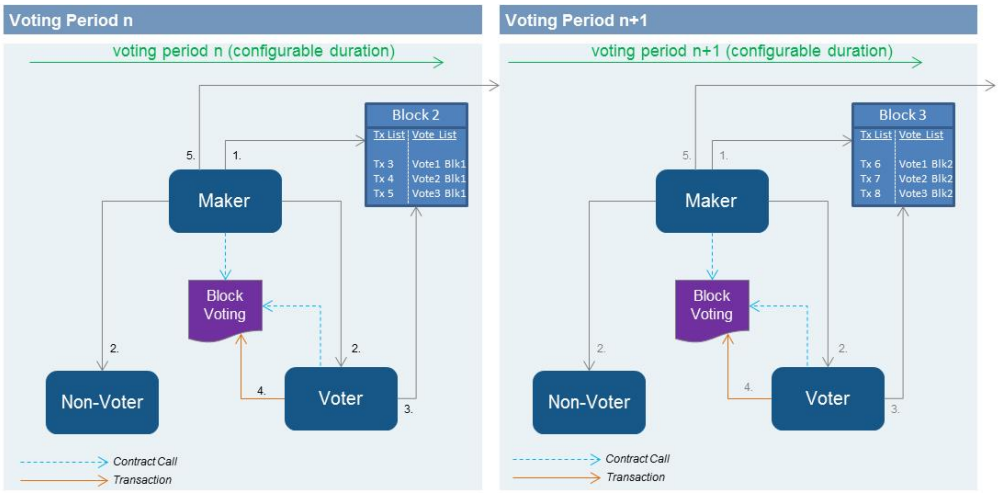

Figura C.1: Flusso del consenso in Quorum

All'interno di un periodo:

1. Il maker node che raggiunge il timeout per primo crea il blocco e lo firma. Il blocco include i voti per il blocco genitore che sono stati trasmessi nel precedente periodo.

2. Il blocco viene pubblicato sulla rete utilzizando lo standard P2P Ethereum Protocol. Tutti i nodi ricevono il blocco, a prescindere dal loro ruolo.

3. I Voters validano il blocco. Questo include:

   (a) Chiamata al contratto BlockVoting per controllare se il maker può effettivamente creare il blocchi (i.e. ha il ruolo maker).

   (b) Chiamata al contratto BlockVoting per controllare se il blocco genitore del blocco in valutazione ha ricevuto abbastanza voti.

(c) Vengono eseguite tutte le transazioni processabili nel blocco ovvero le transazioni pubbliche e quelle private del insieme di partecipanti di cui fa parte il nodo (questo dopo aver recuperato il payload transaciont dal Transaction Manager).

(d) Viene validato lo stato pubblico comparando il *Public state root hash* con lo *State root* all'interno del blocco.

(e) Viene effettuato l'hash di tutte le transazioni del blocco (sia pubbliche che private) e viene quindi confrontato con il *transaction hash* del blocco. Questo viene realizzato per assicurarsi che tutti i Voters concordino sulla lista delle transazioni nel blocco.

4. Una volta validato con successo, i nodi Voters inviano il proprio voto al contratto BlockVoting usando una transazione Ethereum standard che viene distribuita su tutti i nodi. Dato che i voti per un dato blocco sono trasmessi tramite transazioni standard, possono essere processati solo quando il blocco successivo è stato creato.

5. I maker node raggiungono il proprio timeout, determinano se il numero minimo di voti è stato raggiungo per il blocco precedente e poi la procedura di creazione del blocco− >validazione− >votazione viene ripetuta.

Mentre il consenso sullo stato privato è implicito e viene ottenuto attraverso una combinazione sincronizzata di input di contratti (*Global Transaction Hash Validation Check*), un EVM deterministico (*Public State Validation Check*), sincronizzazione della catena (nuovi blocchi aggiunti solo alla catena canonica) è possibile validare ulteriormente il consenso sullo stato privato tramite il comdanto Quorum *eth_StorageRoot*. Questo comando infatti restituisce il *Private State Root Hash* di un account contratto ad un dato numero di blocco che può quindi essere validato rispetto al risultato dello storageRoot di una controparte fuori dalla catena o a livello di applicazione. Questa risulta un'ulteriore forma di sicurezza sulle transazioni private implementata dagli sviluppatori di Quorum.

Appendice D

Script di configurazione

Questo capitolo conterrà tutti gli script di configurazione, utilizzati nell'ambito dell'implementazione dell'applicazione e citati nel capitolo Capitolo 8.

1 Vagrant

Il VagrantFile utilizzato nello sviluppo:

```
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/xenial64"
  config.vm.provision :shell, path: "vagrant/bootstrap.sh"
  config.vm.network "forwarded_port", guest: 22000, host: 22000
  config.vm.network "forwarded_port", guest: 22001, host: 22001
  config.vm.network "forwarded_port", guest: 22002, host: 22002
  config.vm.network "forwarded_port", guest: 22003, host: 22003
  config.vm.network "forwarded_port", guest: 22004, host: 22004
  config.vm.network "forwarded_port", guest: 22005, host: 22005
  config.vm.network "forwarded_port", guest: 22006, host: 22006
  config.vm.provider "virtualbox" do |v|
    v.memory = 4096
  v.gui = false
  end
end
```

Codice D.1: VagrantFile del progetto

Invece il file bootstrap.sh utilizzato per configurare le dipendenze software della blokchain è il seguente:

```
  #!/bin/bash

  set -eu -o pipefail

```

```
5   # install build deps
6   add-apt-repository ppa:ethereum/ethereum
7   apt-get update
8   apt-get install -y build-essential unzip libdb-dev libsodium-
        dev zlib1g-dev libtinfo-dev solc sysvbanner wrk
9
10  # install constellation
11  wget -q https://github.com/jpmorganchase/constellation/
        releases/download/v0.0.1-alpha/ubuntu1604.zip
12  unzip ubuntu1604.zip
13  cp ubuntu1604/constellation-node /usr/local/bin && chmod 0755
        /usr/local/bin/constellation-node
14  cp ubuntu1604/constellation-enclave-keygen /usr/local/bin &&
        chmod 0755 /usr/local/bin/constellation-enclave-keygen
15  rm -rf ubuntu1604.zip ubuntu1604
16
17  # install golang
18  GOREL=go1.7.3.linux-amd64.tar.gz
19  wget -q https://storage.googleapis.com/golang/$GOREL
20  tar xfz $GOREL
21  mv go /usr/local/go
22  rm -f $GOREL
23  PATH=$PATH:/usr/local/go/bin
24  echo 'PATH=$PATH:/usr/local/go/bin' >> /home/ubuntu/.bashrc
25
26  # make/install quorum
27  git clone https://github.com/jpmorganchase/quorum.git
28  pushd quorum >/dev/null
29  git checkout tags/v1.1.0
30  make all
31  cp build/bin/geth /usr/local/bin
32  cp build/bin/bootnode /usr/local/bin
33  popd >/dev/null
34
35  # copy examples
36  cp -r /vagrant/examples /home/ubuntu/quorum-examples
37  chown -R ubuntu:ubuntu /home/ubuntu/quorum /home/ubuntu/quorum
        -examples
```

Codice D.2: Script di bootstrap utilizzato dal Vagrantfile

2 Blockchain Quorum

Il file *init.sh* è stato riorganizzato in maniera tale da avere all'avvio della blockchain due account sul nodo target 1:

```
1   #!/bin/bash
2   set -u
3   set -e
4
5   echo "[*] Cleaning up temporary data directories"
6   rm -rf qdata
7   mkdir -p qdata/logs
```

```
8
9   echo "[*] Configuring node 1"
10  mkdir -p qdata/dd1/keystore
11  cp keys/key1 qdata/dd1/keystore
12  cp keys/key2 qdata/dd1/keystore
13  geth --datadir qdata/dd1 init genesis.json
14
15  echo "[*] Configuring node 2 as block maker and voter"
16  mkdir -p qdata/dd2/keystore
17  cp keys/key2 qdata/dd2/keystore
18  cp keys/key3 qdata/dd2/keystore
19  geth --datadir qdata/dd2 init genesis.json
20
21  echo "[*] Configuring node 3"
22  mkdir -p qdata/dd3/keystore
23  geth --datadir qdata/dd3 init genesis.json
24
25  echo "[*] Configuring node 4 as voter"
26  mkdir -p qdata/dd4/keystore
27  cp keys/key4 qdata/dd4/keystore
28  geth --datadir qdata/dd4 init genesis.json
29
30  echo "[*] Configuring node 5 as voter"
31  mkdir -p qdata/dd5/keystore
32  geth --datadir qdata/dd5 init genesis.json
33
34  echo "[*] Configuring node 6"
35  mkdir -p qdata/dd6/keystore
36  geth --datadir qdata/dd6 init genesis.json
37
38  echo "[*] Configuring node 7"
39  mkdir -p qdata/dd7/keystore
40  geth --datadir qdata/dd7 init genesis.json
```

Codice D.3: Script di inizializzazione della blockchain Quorum

Invece, il file start.sh è stato riscritto per avere i nodi configurati secondo le necessità dell'applicazione:

```
1   #!/bin/bash
2   set -u
3   set -e
4   NETID=87234
5   BOOTNODE_KEYHEX=77
        bd02ffa26e3fb8f324bda24ae588066f1873d95680104de5bc2db9e7b2e510
6   BOOTNODE_ENODE=enode://61077
        a284f5ba7607ab04f33cfde2750d659ad9af962516e159cf6ce708646066cd927a900944ce3
        [127.0.0.1]:33445
7
8   GLOBAL_ARGS="--bootnodes $BOOTNODE_ENODE --networkid $NETID --
        rpc --rpcaddr 0.0.0.0 --rpcapi admin,db,eth,debug,miner,net
        ,shh,txpool,personal,web3,quorum"
9
10  echo "[*] Starting Constellation nodes"
11  nohup constellation-node tm1.conf 2>> qdata/logs/
        constellation1.log &
12  sleep 1
```

```
13  nohup constellation-node tm2.conf 2>> qdata/logs/
        constellation2.log &
14  nohup constellation-node tm3.conf 2>> qdata/logs/
        constellation3.log &
15  nohup constellation-node tm4.conf 2>> qdata/logs/
        constellation4.log &
16  nohup constellation-node tm5.conf 2>> qdata/logs/
        constellation5.log &
17  nohup constellation-node tm6.conf 2>> qdata/logs/
        constellation6.log &
18  nohup constellation-node tm7.conf 2>> qdata/logs/
        constellation7.log &
19
20  echo "Bootnode"
21  nohup bootnode --nodekeyhex "$BOOTNODE_KEYHEX" --addr="
        127.0.0.1:33445" 2>>qdata/logs/bootnode.log &
22  echo "wait for bootnode to start..."
23  sleep 6
24
25  echo "Node 1"
26  PRIVATE_CONFIG=tm1.conf nohup geth --datadir qdata/dd1
        $GLOBAL_ARGS --rpcport 22000 --port 21000 --unlock "0
        xed9d02e382b34818e88b88a309c7fe71e65f419d" --password
        passwords.txt --voteaccount "0x0638e15747$
27
28  echo "Node 2"
29  PRIVATE_CONFIG=tm2.conf nohup geth --datadir qdata/dd2
        $GLOBAL_ARGS --rpcport 22001 --port 21001 --voteaccount "0
        x0fbdc686b912d7722dc86510934589e0aaf3b55a" --votepassword "
        " --blockmakeraccount "0xca8435$
30
31  echo "Node 3"
32  PRIVATE_CONFIG=tm3.conf nohup geth --datadir qdata/dd3
        $GLOBAL_ARGS --rpcport 22002 --port 21002 2>>qdata/logs/3.
        log &
33
34  echo "Node 4"
35  PRIVATE_CONFIG=tm4.conf nohup geth --datadir qdata/dd4
        $GLOBAL_ARGS --rpcport 22003 --port 21003 --voteaccount "0
        x9186eb3d20cbd1f5f992a950d808c4495153abd5" --votepassword "
        " 2>>qdata/logs/4.log &
36
37  echo "Node 5"
38  PRIVATE_CONFIG=tm5.conf nohup geth --datadir qdata/dd5
        $GLOBAL_ARGS --rpcport 22004 --port 21004 2>>qdata/logs/5.
        log &
39
40  echo "Node 6"
41  PRIVATE_CONFIG=tm6.conf nohup geth --datadir qdata/dd6
        $GLOBAL_ARGS --rpcport 22005 --port 21005 2>>qdata/logs/6.
        log &
42
43  echo "Node 7"
44  PRIVATE_CONFIG=tm7.conf nohup geth --datadir qdata/dd7
        $GLOBAL_ARGS --rpcport 22006 --port 21006 2>>qdata/logs/7.
        log &
45
46  echo "Nodes Up. Blockchain up. Run 'geth attach qdata/dd1/geth
        .ipc' to attach to the first Geth node"
```

Codice D.4: Script di avvio della blockchain Quorum

Nella blockchain appena deployata si può notare come ogni nodo possieda un'istanza di geth (ovvero è un full node)

Glossario

Attacco DoS attacco informatico che mira a saturare una o più risorse di rete (inviandogli numerose richieste) con lo scopo, solitamente, di rendere un server incapace di erogare servizi ai propri clienti. Se la richiesta proviene da diversi siti contemporaneamente, si parla di *Attacco DDoS* (Distributed Denial of Service).

Cifratura asimmetrica tecnica di cifratura in cui chiave di crittazione e chiave di decrittazione sono diverse ma complementari, ossia un messaggio cifrato con la prima pu‘ o essere decifrato con la seconda e viceversa; in questo caso ogni utente possiede una coppia di chiavi, una pubblica (nota a tutti) ed una privata (nota solo a se stesso) e per questo, dati N utenti, l'insieme complessivo di chiavi è numericamente pari a *2xN*. È una tecnica di cifratura più sicura rispetto a quella simmetrica, ma allo stesso tempo computazionalmente più complessa.

Cifratura simmetrica tecnica di cifratura tale per cui chiave di crittazione e chiave di decrittazione coincidono, rendendo l'algoritmo molto performante e semplice da implementare; in questo scenario, ogni chiave identifica una coppia di utenti e per questo, dati N utenti, l'insieme complessivo di chiavi definite ha cardinalità: $\binom{N}{2}$.

Dematerializzazione Processo di innovazione tecnologica che prevede la conversione di qualunque documento cartaceo in un adeguato formato digitale, fruibile con mezzi informatici, finalizzata alla distruzione della materialità. Il risultato è una stringa digitale che soddisfa i requisiti tecnici e legali previsti per ciascun tipo di documento elettronico nominato (per esempio, la "ricetta elettronica") o, in termini più estesi, le convenzioni stabilite dalla comunità nella quale il documento assume pieno valore..

EHR-S L'Electronic Health Record – System Functional Model è uno degli standard realizzati da HL7 International ed è approvato, inoltre, da ISO co-

me ISO/HL7 10781. Il modello comprende 322 funzioni e 2.310 criteri di conformità formali..

Hash crittografico algoritmo matematico che mappa dei dati di lunghezza arbitraria (messaggio) in una stringa binaria di dimensione fissa chiamata valore di hash, ma spesso viene indicata anche con il termine inglese message digest (o semplicemente digest). Tale funzione di hash è progettata per essere unidirezionale (one-way), ovvero una funzione difficile da invertire: l'unico modo per ricreare i dati di input dall'output di una funzione di hash ideale è quello di tentare una ricerca di forza-bruta di possibili input per vedere se vi è corrispondenza (match). In alternativa, si potrebbe utilizzare una tabella arcobaleno di hash corrispondenti. La funzione di hash deve avere alcune proprietà fondamentali:

- il calcolo dell'hash deve essere semplice indipendentemente dal dato di partenza.
- deve essere difficile se non impossibile risalire al dato originario dall'hash.
- deve essere improbabile che due messaggi differenti abbiano lo stesso hash

.

Permissioned Ledger Un sistema *permissioned* è quello in cui l'identità per gli utenti è autorizzata nella whitelist (o nella black list); è il metodo comune di gestione dell'identità nella finanza tradizionale. Al contrario, un sistema privo di autorizzazioni è quello in cui l'identità dei partecipanti è caratterizzata da pseudonimi o addirittura anonima. Bitcoin è stato originariamente progettato con parametri privi di permessi..

Pretty Good Privacy (PGP) è una famiglia di software di crittografia per autenticazione e privacy, da cui è derivato lo standard OpenPGP. È probabilmente il crittosistema più adottato al mondo, descritto dal crittografo Bruce Schneier come il modo per arrivare "probabilmente il più vicino alla crittografia di livello militare". Caratteristica principale: robusto ad attacchi MITM (man in the middle). Per esempio con il metodo forza bruta l'attaccante necessiterebbe di un calcolatore con immense capacità di calcolo e di tempi di elaborazioni "universali" (ammesso che il dispositivo che conserva messaggio e chiave privata del mittente non sia compromesso). Funzionamento: il mittente usa la chiave pubblica (cifratura asimmetrica) del destinatario per cifrare una chiave comune (cifratura simmetrica) con cui si cifra il testo in chiaro del messaggio.

Qr-code Un codice QR (abbreviazione di Quick Response Code) è un codice a barre bidimensionale (o codice 2D), composto da moduli neri disposti all'interno di uno schema di forma quadrata. Viene impiegato per memorizzare informazioni generalmente destinate a essere lette tramite un telefono cellulare o uno smartphone. In un solo crittogramma possono essere contenuti fino a 7.089 caratteri numerici o 4.296 alfanumerici. Genericamente il formato matriciale è di 29x29 quadratini e contiene 48 alfanumerici. Il codice è stato sviluppato al fine di permettere una rapida decodifica del suo contenuto.

Ricetta bianca ricetta che il medico compila su carta bianca, sulla quale siano però riportati: il nome e cognome del medico; la data; il luogo; la firma autografa del medico. In questo caso, il nome dell'assistito non è strettamente necessario. Su ricetta bianca possono essere prescritte tutte le prestazioni di specialistica ambulatoriale, di diagnostica strumentale e di laboratorio, di norma correlate alla propria branca di specializzazione e i farmaci, prestazioni che saranno sempre a carico del cittadino assistito. Per la prescrizione a carico del servizio sanitario è infatti necessaria la ricetta del ricettario regionale ed è valida in tutte le farmacie italiane..

Ricetta rossa ricetta che può essere compilata solamente dai medici dipendenti di strutture pubbliche o convenzionati con il servizio sanitario nazionale e viene utilizzata per la prescrizione di una terapia farmacologica, la prescrizione di un esame diagnostico o una visita specialistica a carico del servizio sanitario. L'uso di una ricetta rossa non permette l'erogazione a carico del servizio sanitario di farmaci o prodotti parafarmaceutici non compresi tra le formulazioni del prontuario farmaceutico regionale, né di esami, visite o terapie non comprese nei Lea o nelle disposizioni della propria regione..

Bibliografia

[1] Governo, «Codice dell'amministrazione digitale», *Gazzetta Ufficiale*, vol. 112, 2005, Disponibile presso: http://www.gazzettaufficiale.it/sommario/codici/amministrazioneDigitale (cit. a p. 2).

[2] M. G. e. M. D. Massella Enrica, *La dematerializzazione della documentazione amministrativa*. CNIPA, 2006, Gruppo di Lavoro per la dematerializzazione della documentazione tramite supporto digitale. Disponibile presso: http://www.agid.gov.it/sites/default/files/documenti_indirizzo/libro_bianco_dematerializzazione.pdf (cit. a p. 2).

[3] Governo, «Attuazione dell'agenda digitale italiana», *Gazzetta Ufficiale*, vol. 221, 2012, Disponibile presso: http://www.gazzettaufficiale.it/atto/serie_generale/caricaDettaglioAtto/originario?atto.dataPubblicazioneGazzetta=2012-12-18&atto.codiceRedazionale=12A13277 (cit. a p. 5).

[4] C. dei Ministri, *Disposizioni urgenti per il rilancio dell'economia*. 2013, vol. 69. indirizzo: Disponibilepresso:\url{http://www.gazzettaufficiale.it/eli/id/2013/06/21/13G00116/sg} (cit. alle pp. 5, 7).

[5] G. della Privacy, *Linee guida in tema di Fascicolo sanitario elettronico (Fse) e di dossier sanitario*. 2009. indirizzo: http://www.garanteprivacy.it/garante/doc.jsp?ID=1634116 (cit. a p. 6).

[6] C. dei Ministri, *Regolamento in materia di fascicolo sanitario elettronico*. 2015. indirizzo: http://www.gazzettaufficiale.it/eli/id/2015/11/11/15G00192/sg (cit. a p. 7).

[7] ——, *Agenda e identità digitale. Misure urgenti per la crescita del paese*. 2015. indirizzo: http://www.gazzettaufficiale.it/atto/serie_generale/caricaDettaglioAtto/originario?atto.dataPubblicazioneGazzetta=2012-12-18&atto.codiceRedazionale=12A13277 (cit. a p. 9).

[8] G. R. Marche, «Approvazione del protocollo d'intesa per l'avvio della dematerializzazione della ricetta cartacea», 2014. indirizzo: http://www.cer-sas.it/fpdb/ricetta%20elettronica%20protocollo%20intesa%20DGR0677_14.pdf (cit. a p. 12).

[9] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, Disponibile presso: https://bitcoin.org/bitcoin.pdf, 2008 (cit. a p. 18).

[10] L. Foundation, *Blockchain for business*, Disponibile presso: https://www.hyperledger.org, 2015 (cit. a p. 23).

[11] G. Health, *Gem health for exchanging health data*, Disponibile presso: https://gem.co/health/, 2014 (cit. a p. 23).

[12] G. Becker e R. universität Bochum, *Merkle signature schemes, merkle trees and their cryptanalysis*, Disponibile presso: http://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker_1.pdf, 2008 (cit. a p. 24).

[13] J. O. Diego Ongaro, *In search of an understandable consensus algorithm*, Disponibile presso: https://raft.github.io/raft.pdf, 2014 (cit. a p. 24).

[14] Juno, *Juno conseus algorithm*, Disponibile presso: http://kadena.io/docs/Kadena-ConsensusWhitePaper-Aug2016.pdf, 2014 (cit. a p. 24).

[15] G. Wood, «Ethereum: A secure decentralised generalised transaction ledger», *Cryptopapers*, 2014, Disponibile presso: http://gavwood.com/paper.pdf (cit. a p. 27).

[16] Ethereum, *Ethash*, Disponibile presso https://github.com/ethereum/wiki/wiki/Ethash, Ethereum, 2016 (cit. a p. 35).

[17] J. P. Morgan, *Quorum project*, Disponibile presso: https://www.jpmorgan.com/global/Quorum, 2015 (cit. a p. 37).

[18] B. L. Miguel Castro, «Practical byzantine fault tolerance», *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, 1999, Disponibile presso http://pmg.csail.mit.edu/papers/osdi99.pdf (cit. a p. 39).

[19] D. J. Bernstein, «The security impact of a new cryptographic library», *Academia Road*, vol. 128, 2012, Disponibile presso https://cr.yp.to/highspeed/coolnacl-20120725.pdf (cit. a p. 45).

Ringraziamenti

Chi mi conosce probabilmente non si aspetta una pagina del genere perchè non sono il tipo di persona che esterna spesso quello che pensa. Ma oggi è un giorno speciale e vorrei cercare di ringraziare tutte le persone che mi hanno accompagnato in questo percorso. Se sono arrivato qui è anche grazie alla vostra presenza.

In primis naturalmente *Marianna*, sempre al mio fianco. È difficile spiegare a parole quanto il tuo supporto sia stato determinante in questo cammino. In tutti questi anni hai dovuto sentire le mie lamentele sui problemi più disparati e (secondo me l'hai sempre pensato) più assurdi. Mi hai sempre dato il tuo supporto, il tuo affetto e una parola di conforto. Abbiamo cominciato insieme tanti anni fa e siamo ancora qui. Scusa per tutti i problemi che ti ho creato. Semplicemente grazie, *Amore*.

La mia *famiglia*. Senza di voi non potrei essere qui ed il vostro appoggio è stato costante e solido, pronto ad accettare la distanza e l'assenza da casa del vostro ometto. Mi avete sempre permesso di andare dritto per la mia strada senza dover pensare a null'altro che alla mia carriera, ai miei legami, alle mie distrazioni ed alla mia felicità (anzi quello che si faceva più problemi di tutti ero sempre e solo io).

Una persona che ho incontrato in quest'avventura quasi per caso, ma che è diventata uno dei miei punti di riferimento più grandi. Una vera fortuna conoscerti non saprei come definirla altrimenti. A *Chiara*. Grazie per tutto. E ovviamente non mi posso dimenticare di *Aurora*. Impossibile dividervi. Anche tu mi hai aiutato, anche con 4 semplici chiacchiere vicino al tavolo per svagarmi 5 minuti con il solito "problema al pc", a sentire di meno i peggiori momenti di stress (so che è la prima volta che lo senti dopo tutti questi anni).

Ai miei coinquilini *Fabio*, *Federico*, *Mattia*, *Valerio*. Avete visto il peggio di me e non avete mai avviato una votazione per cacciarmi di casa (il che è tutto dire). Avete condiviso i miei ripassi fino all'ultimo minuto, i miei "e se poi me lo chiede" e, quando tornavo a casa, "alla fine me l'ha chiesto veramente che palle" (lo so che non l'ho mai detto in questo modo ma capitemi).Grazie a tutti voi per aver sempre tentato di farmi ridere con qualche battuta, per le pause caffè e per tutto quello che non sto scrivendo.

A tutti i miei altri coinquilini mancati. *Vincenzo*, *Ilario*, *Giulia*, *Giulia*, *Sara*,

Andrea, *Dante* e a tutte quante le altre persone che per un motivo o per l'altro sono arrivate nella mia vita. Abbiamo condiviso progetti, nottate, spetteguless, caffè, pucce, panini, ansie da pre-esame, ansie da esame, ansie da post-esame, serate in giro (poche). Grazie per tutto.

Al gruppo di Cyber Security dell'UNIVPM, tra i quali sento il dovere di menzionare in particolare (mi permetto di citarvi per nome) i professori Luca, Marco e Franco. Grazie per il costante aiuto, per i consigli e per gli incoraggiamenti in questa opportunità che mi avete concesso.

Grazie infine a tutti voi che siete venuti qui ad ascoltarmi o anche solo per un brindisi, a voi che per un motivo o per l'altro non siete potuti essere qui anche se avreste voluto e a tutti voi che per motivi di spazio non ho citato esplicitamente anche se avrei voluto/dovuto. Grazie.

Ancona, Dicembre 2017 C. P.