Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

AccuMuNet: moderazione di un mercato/sistema CDN basato su peer-to-peer

Laureando

Manuel Zulian

Relatore

Prof. Mauro Migliardi

Co-relatore

Prof. Alessio Merlo

Anno Accademico 2014/2015

Indice

Sommario

Oggetto di questa tesi è lo studio delle problematiche connesse alla realizzazione di un meccanismo di moderazione e garanzia di qualità dei contenuti prodotti da una comunità priva di un sistema centralizzato di controllo. Dopo un'ampia ed approfondita analisi dello stato dell'arte si è deciso di adottare una metodologia basata sulla rete BitTorrent per la distribuzione dei contenuti e sulla block chain di Bitcoin l'autenticazione, ampliando significativamente il lavoro iniziato dall'ing. Andrea Passaglia presso l'Università di Genova. In particolare, la soluzione studiata in questa tesi propone il pieno sfruttamento degli accumulatori crittografici al fine di permettere la gestione di una gerarchia di ruoli con privilegi crescenti.

Capitolo 1

Introduzione

Negli ultimi anni si è assistito a una necessità crescente di decentralizzazione dei servizi web. Questa necessità nasce principalmente come risposta alle richieste di scalabilità in termini di prestazioni [1], e come risposta alle crescenti preoccupazioni sulla privacy che un servizio centralizzato può causare [2].
Per decentralizzazione non si intende solamente una distribuzione del servizio su più entità. Infatti questa non è una condizione sufficiente a garantire la privacy. Possono infatti esistere sistemi distribuiti dotati però di un controllo centralizzato, si pensi come esempio all'ormai defunto Napster, che era dotato di un sistema di coordinamento centralizzato per l'indicizzazione dei file e lasciava ai peer solamente il compito di scambiare i file [3]. L'esistenza di un'entità centrale in un sistema del genere può non solo compromettere la privacy dei singoli utenti, ma introduce un *single point of failure*, come mostra il caso Napster [4]. La tipologia di sistema distribuito più adeguata allo scopo è quella peer-to-peer, in cui tutti partecipano in maniera paritaria a fornire un servizio e le richieste sono risolte localmente o instradate ad altri nodi.
Tuttavia, una delle problematiche più complesse da affrontare quando si definisce un'architettura distribuita di tipo peer-to-peer è rappresentata dalla "moderazione". Si definisce moderazione il processo di eliminazione di contenuti ritenuti inopportuni. In un contesto centralizzato la determinazione di quali contenuti siano inopportuni può essere semplicemente affidata all'entità centrale, in un contesto distribuito invece è più complesso determinare a chi spetti eliminare un contenuto. Infatti, proprio per la natura peer-to-peer dei sistemi in esame, un partecipante è libero di entrare a farne parte alla pari di tutti gli altri [5]. La moderazione in una rete distribuita può riferirsi sia ai partecipanti, sia ai contenuti della rete. Una moderazione a monte dei partecipanti può mitigare a valle il problema della moderazione dei con-

tenuti, rimuovendo il promotore del contenuto considerato inopportuno ed eventualmente invalidando il contenuto stesso.
Due sono gli ambiti principali che si sono studiati in questo lavoro in cui dovrebbe essere applicata una moderazione.
Il primo è quello della pubblicazione dei contenuti. Una delle soluzioni principali per il problema della scalabilità delle prestazioni nei servizi web è attualmente l'uso di una rete CDN (Content Delivery Network). Queste reti sono dei proxy distribuiti, anche geograficamente, che hanno il compito di mitigare gli attacchi DDoS e migliorare la raggiungibilità dei contenuti. Le soluzioni commerciali più famose sono attualmente centralizzate, ma si stanno studiando metodi di natura peer-to-peer e in questo senso una moderazione dei contenuti serviti dai peer è indispensabile [6].
Il secondo ambito studiato in cui è necessario applicare una moderazione è quello dei marketplace distribuiti. Il più famoso marketplace centralizzato elettronico è eBay. Negli ultimi anni sono però emersi tentativi di realizzare marketplace distribuiti in cui non esiste un'unica entità centralizzata a dirimere le questioni tra acquirente e venditore. Uno dei più riusciti di questi tentativi è OpenBazaar [7]. Chiaramente in un contesto come quello dello scambio commerciale la moderazione dei partecipanti e dei contenuti è una necessità sentita.
In generale sono state esaminate diverse tecnologie per raggiungere l'obiettivo. Per prima cosa si sono cercate soluzioni adatte allo scopo nell'ambito delle reti CDN, poiché essendo storicamente centralizzate si pensava avessero proposto soluzioni per la moderazione anche in ambito distribuito. Poi si è proseguito ad esaminare altre soluzioni in ambito puramente distribuito, in particolare OpenBazaar e Zeronet. La soluzione ritenuta più promettente è stata quella di Twister, un social network distribuito che basa il suo funzionamento sulle tecnologie BitTorrent e Bitcoin. L'obiettivo di Twister, tuttavia, è quello di creare un clone distribuito di Twitter, quindi non è direttamente impiegabile per lo scopo prefissato. Tuttavia esiste un'implementazione delle stesse tecnologie che ha lo scopo di affrontare e risolvere il problema della moderazione in ambito distribuito. Si tratta di Accumunet, software sviluppato in una precedente tesi dall'ing. Andrea Passaglia all'Università di Genova.
Per capire come funziona Accumunet bisogna prima descrivere il funzionamento di Twister. Twister unisce due tecnologie, la block chain di Bitcoin [8] e la rete BitTorrent con relativa rete DHT [8], per raggiungere l'obbiettivo di creare un clone di Twitter altamente anonimizzato e sicuro [8]. La rete block chain viene usata per memorizzare i nomi utente e le corrispondenti chiavi pubbliche, alterando il significato originale delle transazioni Bitcoin [8]. In questo schema un utente può farsi riconoscere firmando i propri messaggi, e

quindi diventa fondamentale per l'utente conservare la propria chiave privata.
La rete BitTorrent invece viene usata per memorizzare qualunque altro tipo di dato che non vada inserito nella block chain, per evitare di inquinarla con informazioni non strettamente fondamentali. In particolare la rete BitTorrent è usata per lo scambio di messaggi diretto tra utenti e per aggiornare gli utenti della presenza di nuovi messaggi nella bacheca.
Accumunet propone di risolvere il problema della moderazione memorizzando nella block chain un accumulatore crittografico. Si tratta di un costrutto che, in una delle sue implementazioni più efficienti in termini di spazio occupato, permette di autenticare l'appartenenza di un'entità a un gruppo specifico. Questo gruppo specifico non è altro che il gruppo a cui sono assegnati i privilegi di moderazione. Questo schema può essere generalizzato a più gruppi nella stessa rete, ognuno con privilegi diversi.
La soluzione di Accumunet al problema della presentazione dei contenuti è invece una generalizzazione di quelli che in Twister sono i post in bacheca.

I punti di Accumunet che si andranno a elaborare sono due, entrambi attribuili alla natura prototipale del lavoro:

- **Punto di accesso unico per tutti i fornitori di contenuti** In Accumunet, per vedere i contenuti di un fornitore è necessario iscriversi alla sua lista di prodotti manualmente, meccanismo che ricalca in parallelo l'iscrizione a un feed di messaggi nell'ambito di un social network. Bisogna tuttavia prima scoprire l'esistenza di questo fornitore, presumibilmente per vie parallele. La soluzione da me proposta invece sarà quella di una struttura unica, i cui parametri fondamentali e la cui distribuzione dei contenuti sarà memorizzata tramite una soluzione a block chain. In questa struttura sono memorizzati anche quali fornitori devono comparire e in quale ordine.

- **Meccanismo di variazione dell'accumulatore** In Accumunet l'accumulatore crittografico era fisso, inserito al momento di avvio del sistema nella block chain e non più modificabile. In un contesto reale si presenta la necessità di aggiornare questo costrutto, aggiungendo o rimuovendo fornitori a seconda della situazione. La soluzione adottata per l'aggiornamento prende diretta ispirazione dalle *multisignatures* di Bitcoin, pur non implementandole direttamente.

Per prima cosa verranno presentate le soluzioni esaminate, per poi passare a una versione migliorata di questo sistema, denominata CoMoNet

(Content Moderated Newtork). Infine si esamineranno possibili estensioni e miglioramenti a questo sistema.

1.0.1 Possibile scenario d'uso

Mentre si prendevano in considerazione i marketplace distribuiti, si è individuato un possibile caso d'uso di una tecnologia che permettesse la moderazione in ambito distribuito. Si tratta di risolvere un problema per i piccoli commercianti, ovvero quello della gestione di un portale dove poter presentare i propri prodotti in maniera autonoma e autogestita. Si vuole infatti evitare una soluzione che richieda un finanziamento continuo, che per sua natura finirebbe per essere trascurata e dimenticata. Si pensi come esempio a un portale centralizzato creato tramite appalto: questo richiederebbe un costo continuo in termini di banda, registrazione del dominio, energia elettrica, gestione degli aggiornamenti dei contenuti. La soluzione oggetto di questa tesi propone di distribuire i costi di mantenimento, pur ammettendo eventualmente un'intervento centralizzato e coordinato iniziale per far partire la struttura.

Capitolo 2

Analisi dello stato dell'arte

2.1 Content delivery networks

Per prima cosa si è studiato lo stato attuale delle reti per la distribuzione dei contenuti. Tradizionalmente la denominazione CDN viene usata per indicare le grosse reti commerciali per la distribuzione dei contenuti gestite da varie aziende, come per esempio Akamai, Amazon, CloudFlare e altri[6]. In questa analisi invece l'attenzione ci concentrerà sulle distribuzione di contenuti nel senso più ampio del termine.
In particolare si analizzeranno reti P2P gestibili dagli utenti, in quanto l'obiettivo è quello di realizzare un sistema che sia autonomo e auto-gestito dai partecipanti. Si analizzerà comunque un approccio ibrido P2P-CDN classica, che può dare degli spunti in ambito commerciale e mobile.
Non si sono trovati studi riguardo la moderazione dei contenuti su reti distribuite. Solo l'ultimo approccio ibrido tra una rete P2P e una CDN classica sembra lasciare spazio a qualche idea sull'argomento [6].

Per quanto riguarda le reti puramente P2P, queste si dividono in due grosse categorie:

1. **Non strutturate** Si intendono reti che non creano nessuna struttura tra i peer che vi partecipano, ammettendo la formazione di connessioni casuali tra loro. Questo tipo di approccio facilita la creazione della rete, ma penalizza le operazioni su di essa, ad esempio la ricerca, che non possono avvalersi di informazioni aggiuntive sulla disposizione dei nodi e dei contenuti come invece potrebbe avvenire in presenza di una struttura ben definita.
   Le reti di questo tipo possono essere centralizzate, decentralizzate o ibride. La differenza sta nella presenza o meno di un server che tiene un indice dei peer collegati e dell'appartenenza di un file a un certo peer

o meno. Le reti non strutturate sono sistemi particolarmente adatti al file-sharing, ma non allo scopo di questa tesi. Infatti il vantaggio di questi sistemi è una ricerca di file tramite keyword semplificata, a scapito però delle prestazioni di questa ricerca. I classici metodi di ricerca dei contenuti su queste reti, BFS, Iterative search e Random Walk [10], hanno infatti prestazioni $O(n)$ [10]. Questi metodi, come già accennato sopra, generano una grande quantità di overhead, e a volte non hanno nemmeno la garanzia di trovare il file se presente. Si può ipotizzare che nel sistema in oggetto di questa tesi i contenuti siano relativamente statici, sicuramente più di quanto richiesto da un sistema di file-sharing dinamico. Infatti il sistema qui implementato cerca di sostituire un portale centralizzato per l'esposizione di prodotti, per la descrizione di eventi e possibilmente anche per la vendita, in cui i contenuti non variano con frequenza elevatissima.

2. **Strutturate** Questa tipologia di reti istituisce una struttura sui peer per semplificare e velocizzare la ricerca. Il trade-off si ha nell'aumento del costo di mantenimento della struttura. Esempi tipici di struttura sono quella ad albero, ipercubo, anello e variazioni ibride tra queste. Si ha quindi un vantaggio nell'utilizzo di queste reti solamente se il *churn-rate*[1] è basso. É questo il caso del sistema qui in esame: si suppone che i fornitori di contenuti abbiano interesse a partecipare alla rete più attivamente possibile, per aumentare la propria presenza. Si può quindi pensare che idealmente un fornitore di contenuti cerchi di restare all'interno della rete il più tempo possibile.

Le reti CDN adottano poi delle politiche di replicazione dei contenuti da cui il sistema in esame può trarre beneficio [10]:

1. **Replicazione passiva** Si parla di replicazione passiva quando un contenuto viene replicato su più peer a causa della sua popolarità, seguendo la domanda. É possibile che questo meccanismo possa essere implementato nel sistema in esame: un prodotto particolarmente popolare potrebbe essere replicato sia dai fornitori di contenuti sia dagli utenti stessi nel caso in cui questi diventino parte attiva della rete. In questo caso e' richiesto, ovviamente, un qualche incentivo, come sconti o promozioni. Questo meccanismo richiede tuttavia il raggiungimento di una certa massa critica di utilizzatori, sia di fornitori di contenuti sia di utenti.

[1] Con *churn-rate* si intende il tasso di utenti che entrano e lasciano il sistema in determinato periodo. Se elevato, questo costringe la struttura ad aggiornarsi troppo spesso e la rete perde la maggior parte del tempo e spreca traffico di rete a mantenere la struttura.

2. **Replicazione attiva** Si parla di replicazione attiva quando la distribuzione dei contenuti è pianificata a priori e non guidata dalla domanda. Si può ipotizzare di implementare sin da subito un meccanismo di questo tipo per risolvere problemi di raggiungibilità di un fornitore di contenuti, per motivi tecnici magari.

Si può inoltre tentare un approccio ibrido tra una costruzione P2P e una CDN gestita da un ente centralizzato, approccio che detto PAC o CAP a seconda dell'implementazione [6]. I due acronimi stanno rispettivamente per "Peer-aided CDN" e "CDN-aided P2P". Il primo metodo consiste nella compresenza di una rete P2P di utenti e di una rete CDN di distribuzione gestita da un ente terzo. Un utente può inoltrare la richiesta a un nodo di frontiera della CDN, a quel punto la CDN può decidere di servire direttamente il contenuto se il server ne è in possesso, inoltrare la richiesta alla CDN o redirigere l'utente a un peer in possesso del contenuto. Il secondo metodo invece è più incentrato sul P2P, e la rete CDN viene usata solo come backup, bootstrapping o comunque come coadiuvante. Questo approccio permette di conservare un certo livello di centralizzazione. Nel nostro schema, è ipotizzabile che i fornitori di contenuti si riuniscano sotto un'associazione, magari diretta da un presidente, che mette a disposizione le risorse verso una rete esterna. Anticipando quello che verrà detto dopo, se il mantenimento del sistema richiede la gestione di una block chain, è difficile che un utente decida di mantenere una block chain solamente per vedere i contenuti di un fornitore. Le alternative sono quindi due:

1. Si prevede un meccanismo di premio per i partecipanti alla rete. In particolare, si può pensare di assegnare una posizione centrale in home page ai fornitori di contenuti che scoprono nuovi blocchi nella block chain, e di assegnare agevolazioni nella fruizione di questi contenuti agli utenti che fanno altrettanto.

2. Si fornisce all'utenza un client "stupido", che non fa altro che trattare i partecipanti alla rete come server, senza mantenere una block chain in locale. Ecco che i fornitori di contenuti che mantengono la rete assumono il ruolo della CDN nell'approccio PAC descritto in precedenza, e gli utenti possono al limite assumere il ruolo dei peer. É importante notare che questo potrebbe essere l'unico approccio possibile in ambito mobile, dove il mantenimento di una block chain è proibitivo in termini di consumo e spazio.

Per quanto riguarda l'ultimo approccio esaminato, l'adozione di una rete separata per i consumatori e per i fornitori di contenuti potrebbe permettere

una gestione differente per la rete di quest'ultimi. La ricerca di come farlo però ha motivato l'esame di un'altra categoria di reti, più specifiche per la gestione della compravendita di prodotti e quindi più probabilmente dotate di meccanismi in grado di controllare i contenuti.

2.2 Marketplace distribuiti

L'esame di questa categoria di reti/prodotti è motivato dalla volontà di creare un portale distribuito auto-gestito dai fornitori di contenuti, senza la necessità di un elemento centralizzante. In realtà l'obbiettivo qui non è di creare un mercato o un luogo dove abbiano luogo transazioni di denaro, ciò non toglie che questa possa essere una naturale estensione del sistema. In particolare la soluzione adottata alla fine impiega la tecnologia Bitcoin per scopi non strettamente finanziari, tuttavia nulla vieta di sfruttarla anche per il suo significato originale, cioè lo scambio monetario.

2.2.1 OpenBazaar

OpenBazaar si definisce come un mercato decentralizzato peer-to-peer in cui i beni vengono scambiati mediante Bitcoin. Si tratta di un fork di Dark Market, un progetto vincitore di un Hackaton al Bitcoin Expo di Toronto nell'Aprile 2014 e sviluppato in risposta alla chiusura di Silk Road[2] [11].
Il progetto viene descritto come l'incontro tra eBay e BitTorrent. La tecnologia alla base degli scambi tra utenti infatti è una DHT derivata da Kademlia che condivide il suo funzionamento con quella di BitTorrent [11] [7]. Gli utenti si scambiano tra loro non i prodotti direttamente, ma quelli che sono chiamati Contratti Riccardiani [14]. Questi contratti sono dei file in cui è contenuto l'identificativo del venditore, l'indirizzo Bitcoin del venditore, i dati della merce o del servizio in vendita, la chiave pubblica PGP del venditore e la firma di tutto il contratto con la corrispondente chiave privata. Il formato di questi file dipende dall'implementazione, due possibili esempi sono il formato json e il formato xml. Per capire la funzione dei campi di un contratto è fondamentale descrivere l'interazione con cui avviene uno scambio sulla rete.
Per prima cosa, bisogna specificare che ogni peer sulla rete può assumere uno tra tre ruoli durante una transazione: venditore, acquirente o arbitro. Quest'ultimo è un'entità incaricata di risolvere le dispute tra i primi due in

[2]Silk Road era un mercato centralizzato di scambio di beni, per la gran parte illegali, che operava sulla rete anonima Tor. Operativo dal 2011 al 2013, anno in cui è stato chiuso dall'FBI.

caso di necessità. Una transazione avviene mediante attraverso i seguenti passi [12]:

1. Il venditore firma il proprio contratto e lo rende disponibile attraverso la rete DHT.

2. L'acquirente recupera il contratto e lo firma a sua volta per dichiarare la sua intenzione ad acquistare. L'acquirente manda a questo punto il contratto a un arbitro.

3. L'arbitro firma a sua volta il contratto, inserendo la sua chiave pubblica PGP, il suo identificativo e la sua chiave pubblica Bitcoin. Genera inoltre un indirizzo Bitcoin multisignature di tipo 2-of-3 dalle tre chiavi accumulate fino a questo punto, e il corrispondente *redeem script*. Quest'ultimo è una sequenza di istruzioni scritte nel linguaggio a stack incluso in Bitcoin. Il risultato dell'esecuzione di queste istruzioni determina se la transazione può essere spesa o meno. Operazioni tipiche di questo linguaggio sono l'hash con algoritmo SHA-256, il confronto di elementi sullo stack, la duplicazione di elementi sullo stack, la verifica di firme e altre ancora [13]. Alla fine di questo punto il contratto è inoltrato a tutti i partecipanti alla transazione.

4. L'acquirente invia i Bitcoin necessari per l'acquisto all'indirizzo multisignature.

5. Il venditore spedisce il prodotto o rilascia il servizio.

6. Al ricevimento del bene, l'acquirente crea una transazione per lo sblocco dei fondi all'indirizzo multisignature con la sua parte della firma.

Nel caso l'acquirente si rifiutasse di pagare all'ultimo punto, può intervenire l'arbitro, firmando con la sua chiave a favore del venditore. Può anche accadere il contrario, ovvero che il bene non sia conforme a quanto dichiarato dal venditore. In tal caso l'arbitro e l'acquirente firmeranno entrambi per sbloccare la transazione multisignature a favore dell'acquirente.
Tutto questo meccanismo introduce un problema fondamentale nel sistema, cioè la fiducia da riporre nell'entità dell'arbitro. Un tentativo di soluzione a questo problema è l'introduzione di una particolare versione di Web of Trust [14].
Una Web of trust è un grafo diretto tra due entità che rappresenta la fiducia che ha la prima nei confronti della seconda, e non è una relazione simmetrica. Una delle caratteristiche principali di OpenBazaar è la possibilità di

restare anonimi, e questo viene tradotto nella Web of Trust con il meccanismo dello pseudonimo. Ogni utente è identificato da un nome user-friendly, che è associato all'hash di una chiave pubblica generata dall'utente grazie al supporto di NameCoin [14]. La soluzione più semplice, ma non adottabile in questo caso, è un rating system globale ispirato direttamente a eBay. Questa soluzione non è adottabile a causa della natura decentralizzata del sistema, che la renderebbe particolarmente vulnerabile ai *Sybil Attacks* [15] [16]. La soluzione in questo caso passa dalla combinazione delle seguenti due strategie.

Partial trust

Questa strategia si basa sull'idea che una visione globale della rete per un nodo potrebbe compromettere l'anonimato. Per evitare questo problema, ogni nodo assegna un valore di fiducia solo ai nodi direttamente collegati a lui. Quando deve fare una transazione con un nodo con cui non è direttamente in contatto, può chiedere la fiducia ai suoi vicini, i quali possono eventualmente ripetere il procedimento. A ogni passaggio interviene però un fattore di smorzamento che diminuisce la fiducia indiretta. Questo sistema si basa sulla transitività della fiducia.
É importante notare che questo schema è vulnerabile ad un attacco di tipo man-in-the-middle. Un entità malevola può impersonare un venditore replicandone la lista di prodotti, ricevendo le richieste e inoltrandole al reale venditore, facendo da tramite. Al termine della transazione, l'attaccante riceve fiducia positiva sia dal venditore sia dall'acquirente. L'unico modo per evitare questo tipo di attacco è quello di verificare l'identità del venditore con altri mezzi prima di effettuare la transazione.
Si può ipotizzare un'integrazione con altri tipi di Web of Trust, come ad esempio GPG, ma generalmente un'associazione di questo tipo è da evitare perché può causare problemi all'anonimato (paradossalmente, si potrebbe integrare il sistema con un social network come Facebook, al prezzo però di distruggere l'anonimato).

Global trust

L'idea alla base della fiducia globale prende ispirazione dal mondo reale: quando un negozio apre i battenti, è presumibile rimarrà aperto per un lungo periodo per ammortizzare i costi di apertura, e non sparirà dalla circolazione il giorno dopo. Il venditore effettua quindi un investimento che è anche una garanzia della sua affidabilità. Questo tipo di fiducia può servire a fare del bootstrapping per un venditore che ha appena iniziato la sua attività, o

comunque per un entità che vuole presentarsi come garantita a prescindere dalle relazioni che ha con altri nodi. Ci sono 4 possibili tecniche utilizzabili per raggiungere questo obiettivo, di cui due sono attaccabili e due sono più sicure:

- **Proof of donation** Per garantire la propria onestà, il venditore impegna il proprio denaro nei confronti di un'associazione benefica. Tecnicamente questo è realizzato tramite una transazione Bitcoin che ha l'organizzazione come destinatario e contiene l'identificativo del venditore. Questo tipo di prova è vulnerabile ad un attacco nel caso il venditore riesca a ottenere le chiavi private dell'organizzazione (o più semplicemente crei un'organizzazione fittizia): è sufficiente riportare il denaro al venditore per aumentare in maniera arbitraria la fiducia globale.

- **Proof to miner** Simile allo schema precedente, ma il denaro è assegnato al primo miner che include la transazione in un nuovo blocco nella rete Bitcoin. Tecnicamente è sufficiente realizzare transazioni con script di output che contiene solo l'operatore OP_TRUE, oppure con zero output, in quel caso tutto l'input è assegnato come fee al miner. L'attacco in questo caso può essere portato dal miner stesso: è sufficiente che realizzi una transazione che tiene segreta fino a che non riesce a generare un blocco. A quel punto la inserisce nel blocco e guadagna un ammontare arbitrario di fiducia globale.

- **Proof of burn** Questo schema prevede invece di "bruciare" del denaro, creando transazioni con l'operatore OP_FALSE nello script di output, e quindi inspendibili. Questa tecnica è più sicura delle due elencate precedentemente e previene anche i Sybil Attacks, perchè sarebbe troppo costoso creare tante entità con molta fiducia.

- **Proof of timelock** L'idea di questo schema è di bloccare il denaro per un certo periodo di tempo al posto di renderlo non spendibile. Dal punto di vista psicologico è l'approccio migliore, tuttavia è tecnicamente difficile da realizzare sulla rete Bitcoin. Le transazioni hanno infatti un campo *nLockTime* che però viene ignorato dai nodi. Non è escluso però che questo approccio sia implementabile in altre reti.

Alla fine la fiducia totale nei confronti di un nodo è la combinazione di quella parziale e quella totale con una formula che può variare a seconda di criteri per lo più empirici.
Nonostante il sistema in generale sia adatto a quello che si voleva fare, si

è trovato difficile introdurre da zero un elemento di moderazione in questo sistema. In particolare non c'è nessuna block chain da sfruttare integrata, e si dovrebbe ricorrere a sistemi esterni come Namecoin. Il punto principale però è che questo sistema è stato pensato per essere il più decentralizzato possibile, e questo va in contrasto con il requisito di avere un gruppo che modera la rete che è l'obbiettivo di questo elaborato.

2.2.2 ZeroNet

ZeroNet è un progetto molto giovane, pubblicato su GitHub la prima volta a gennaio del 2015. Il progetto prevede di servire siti web su rete BitTorrent. L'idea è che ogni utente replica automaticamente i siti che visita, diventando un peer per quel sito [17]. La creazione di un sito ZeroNet inizia con la generazione di una coppia di chiavi pubblica/privata, usando lo stesso algoritmo di generazione delle chiavi usato da Bitcoin (ECDSA). La chiave pubblica, che funziona anche da indirizzo Bitcoin, è l'indirizzo del sito. La chiave privata viene invece usata per firmare i contenuti e pubblicare gli aggiornamenti al sito. Quando un utente vuole visitare una pagina, per prima cosa chiede al tracker una lista di peer che servono il contenuto interessato. Il tracker registra inoltre l'utente come nuovo peer che può servire quel contenuto. Per completezza, è opportuno osservare che è supportata anche una distribuzione trackerless dei contenuti.

Una volta recuperata una lista dei peer, il client chiede a uno di questi il file content.json, che contiene la struttura del sito. Questo file è firmato dal proprietario del sito: quando il client lo riceve, ne verifica l'autenticità con l'indirizzo del sito, che altro non è che la chiave pubblica corrispondente. Una volta verificato il file, il client può procedere a scaricare i contenuti indicati in questo, in particolare la pagine html, i file .css e i file .js. Nel listato 2.1 è possibile vedere un esempio di questa struttura. Tutti i file sono accompagnati dal loro hash SHA-512, per garantire che non siano oggetto di falsificazione da parte di un attaccante. Il sistema prevede un meccanismo di modifica dei siti. Per prima cosa il proprietario firma un nuovo file content.json, poi procede a inoltrarlo ai visitatori del sito, che verificano di avere l'ultima versione disponibile. É interessante notare che gli aggiornamenti sono in tempo reale, e vengono propagati tramite la tecnologia WebSocket del browser.

```
1  {
2    "address": "1Name2NXVi1RDPDgf5617UoW7xA6YrhM9F",
3    "title": "ZeroName",
4    "description": "Namecoin address registry",
5    "files": {
6      "css/all.css": {
7        "sha512": "f00818c5b52013a467dc1883214b57cf6ac3dbe6da2d
       f3f0af3cb232cd74877b",
8        "size": 69952
9      },
10     "data/names.json": {
11       "sha512": "341e4b1eb28a9aebef1ff86c981288b7531ec957552c
```

```
      f9a675c631d1797a48df",
12        "size": 1002
13      },
14      "index.html": {
15        "sha512": "b3fd5f2e61666874b06cc08150144015c0e88c45d3
      e7847ff8d4c641e789807d",
16        "size": 2160
17      },
18      "js/all.js": {
19        "sha512": "4426ca2dfacd524fb995c9f7522ca4e6f70c3e524b4b
      d8ca67f6416f93fca111",
20        "size": 90523
21      }
22    },
23    "signers_sign": "HOKZByY9pO2Iqh5UE+Nb7N5qb2cTvhULB3euvszufD
      nGIVeF4mswur3PyXxGXM+tJ8kZOFzspFRIl0gOyCE0tCM=",
24    "signs": {
25      "1Name2NXVi1RDPDgf5617UoW7xA6YrhM9F": "G6X42ZmEBf66jjylSn
      x45Uee9J+QO7dLt1CLYULI17L78AFaUDVHYohEYUGxAFqKx75UpWGsPGS
      Y1S7lr/Fe3EU="
26    },
27    "signs_required": 1,
28    "ignore": "(js|css)/(?!all.(js|css))",
29    "modified": 1429483269.681872,
30    "zeronet_version": "0.2.9"
31  }
```

Listing 2.1: Esempio di struttura di un sito Zeronet.

Il sistema è estremamente interessante per lo scopo di questa tesi, visto anche il supporto per i siti multi-utente. Il tipico esempio d'uso è quello di un forum. Nel caso un utente volesse postare, comunica al proprietario del forum una chiave pubblica con cui firmare i propri contenuti. Quindi modifica il file content.json per permettere ai visitatori di ricevere aggiornamenti direttamente dal nuovo utente, senza richiedere l'intervento del proprietario stesso ogni volta. Resta ferma la possibilità da parte del proprietario di rimuovere la possibilità di postare per ogni utente, permettendo un controllo contro lo spam e gli abusi. Anticipando quella che sarà la soluzione poi adottata da CoMoNet, si può ipotizzare di usare un accumulatore crittografico all'interno del file content.json per tenere traccia di tutti gli utenti che possono postare, a condizione che il client verifichi ogni volta se l'utente è accumulato o meno quando posta.

Si è scelto comunque di non costruire su questa soluzione perché, nonostante il progetto proclami di utilizzare la tecnologia di crittografia di Bitcoin, non c'è nessuna block chain sottostante e la crittografia viene usata solamente

nella generazione degli indirizzi [18]. Questo non permette di fare affidamento sulle garanzie di autenticazione fornite da una block chain. É opportuno osservare tuttavia che ZeroNet usa NameCoin per registrare i domini .bit, realizzando così in un qualche modo la stessa associazione etichetta-chiave pubblica che si ritrova in CoMoNet sotto forma della coppia nome utente-chiave pubblica. Si è comunque scelto di non adottare questa soluzione perché un controllo autonomo della block chain permette una migliore gestione delle politiche di aggiornamento dei dati sensibili come gli accumulatori, e in generale permette più flessibilità. Nulla però vieterebbe a priori di usare NameCoin per memorizzare gli accumulatori stessi e usare quella block chain al posto di quella di CoMoNet.

2.2.3 Accumunet

Accumunet nasce come fork di Twister nel 2014, da un lavoro di tesi di Andrea Passaglia all'Università di Genova [9]. Twister è una piattaforma di micro-blogging peer-to-peer, il cui obbiettivo è quello di creare una versione decentralizzata di Twitter per evitare problemi legati alla censura [19]. La motivazione fondamentale di Accumunet era la creazione di un mercato distribuito ma che contenesse anche un elemento di moderazione, il che ne fa la base ideale per l'obiettivo di questo lavoro. Il suo funzionamento è fondamentalmente inestricabile da quello di Twister, per cui verrà descritto quest'ultimo inserendo le modifiche apportate da Accumunet lungo la descrizione.

Funzionamento di Twister

Prima di descrivere il funzionamento di Accumunet, è indispensabile esaminare il funzionamento di Twister, su cui si basa. Lo schema strutturale di Twister è mostrato in figura 2.1.

Bitcoin

La parte principale di Twister è un derivato dell'originale client Bitcoin, detto anche client Satoshi. Questo componente ha tre funzioni principali:

- Mantenere la blockchain con i dati di autenticazione degli utenti.
- Avviare la rete bittorrent e la rete dht.
- Esporre un'interfaccia di utilizzo all'utente mediante una api RPC-JSON.

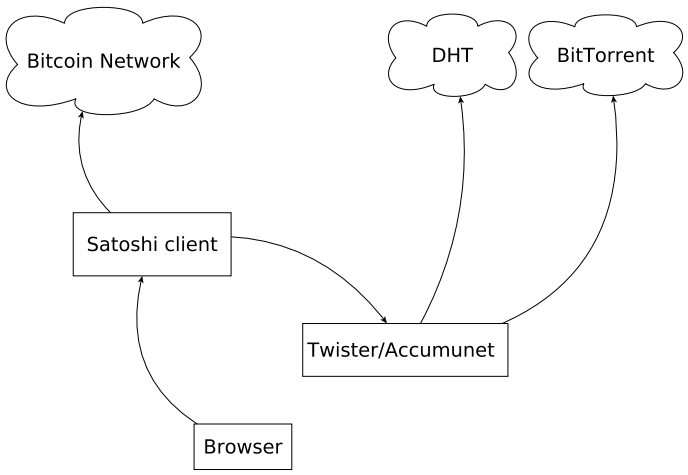

Figura 2.1: Struttura di Twister/Accumunet

Bitcoin nasce nel 2008 dal lavoro di Satoshi Nakamoto [20], pseudonimo adottato da uno o più programmatori di cui non si è attualmente scoperta l'identità. É il tentativo di moneta elettronica decentralizzata di maggior successo, e risolve il problema del double spending adottando diverse soluzioni piuttosto avanzate dal punto di vista tecnico.
L'intero sistema si basa su transazioni tra utenti. Si supponga per il momento che gli utenti siano già in possesso di bitcoin, in un modo o nell'altro. Si supponga inoltre che l'utente Alice voglia inviare una somma di denaro all'utente Bob. Per prima cosa, Alice recupera tutte le transazioni dirette a lei dalla block chain (che verrà descritta in seguito). A questo punto, con la lista di tutti i bitcoin in suo possesso, può creare una nuova transazione collegandone gli input agli output di transazioni precedenti in suo possesso. Alice poi inserisce nell'output della transazione l'indirizzo di Bob e uno script che dice essenzialmente che l'unico modo per sbloccare e spendere quel denaro è quello di presentare una firma della transazione con la chiave privata di Bob. A questo punto Alice invia la transazione agli altri peer nella rete e il pagamento si può considerare "effettuato", almeno per quanto riguarda Alice. Prima di descrivere il resto del funzionamento, alcune osservazioni:

- Una transazione può avere più di un input e più di un output, questo permette di ricreare quello che nel mondo reale è il sistema del "cambio" nei pagamenti. In questo caso il cambio è essenzialmente un output che ritorna parte dell'input al proprietario.

- Input e output non sono uguali, e il motivo sarà descritto in seguito quando verrà presentata la block chain.

- Ogni utente possiede una serie di coppie di chiavi pubbliche e private generate con crittografia a curva ellittica. Per evitare di rendere identificabili i protagonisti di una transazione è buona norma usare un indirizzo bitcoin diverso per ogni transazione, indirizzo generato tramite hashing della chiave pubblica.

- Per specificare da chi può essere speso un output, Bitcoin implementa un meccanismo di scripting basato su stack che però non è Touring-completo. Questo permette una certa flessibilità nell'uso delle transazioni, come si può vedere per esempio nell'implementazione delle multisignature, transazioni il cui output può essere speso solo presentando la firma di almeno m tra n utenti.

Il secondo componente del sistema Bitcoin, e forse quello pi importante, è quello che risolve il problema del double-spending, cioè la block chain. Questa, come dice il nome, è una catena di blocchi, dove i blocchi contengono transazioni di tutti gli utenti. Quello che succede quando Alice invia il pagamento è che inoltra la transazione nella rete. La transazione viene quindi raccolta dai cosiddetti miner, ovvero peer che lavorano alla generazione di nuovi blocchi. I miner raccolgono le transazioni e le inseriscono nel blocco che stanno cercando di generare. Per la generazione di un blocco, bisogna generare una proof-of-work. Si prendono le transazioni raccolte, l'hash del blocco precedente e un nonce, e si varia il nonce fino a che l'hash di tutto non contiene un numero sufficiente di zeri all'inizio, numero definito target. Il target è gestito dinamicamente nella rete Bitcoin in modo tale che servano più o meno dieci minuti per generare un nuovo blocco. Variando il numero di zeri all'inizio, la rete può aggiustare la difficoltà di generazione di un nuovo blocco e quindi il tempo.
Questo meccanismo risolve il problema del double spending. É infatti possibile che un utente provi a spendere l'output di una transazione più volte. In questo caso però si può verificare che l'output non sia già stato speso guardando nella block chain. Un attaccante che volesse riscrivere la block chain a suo favore, dovrebbe rigenerare tutti i blocchi dalla transazione che vuole cambiare in poi e farli accettare dalla rete. Grazie alla proof-of-work, questo è computazionalmente infattibile. É buona norma però aspettare un numero adeguato di blocchi (in genere 6) prima di accettare una transazione come confermata.
Resta un problema, ovvero quello di incentivare i peer nella rete a lavorare come miner. Questo si risolve in due modi:

- All'inizio grazie alla cosiddetta transazione coinbase. Quando un miner scopre un nuovo blocco, è autorizzato inserire nel blocco una transazio-

ne con nessun input e un output indirizzato a lui di un certo numero di bitcoin. Questo numero cala nel tempo, perchè il numero di bitcoin è limitato a 21 milioni[3].

- A regime nella rete (e quando i bitcoin saranno stati tutti generati) con i fee delle transazioni. Quando prima si accennava che input e output nella transazione non sono quasi mai uguali, si intendeva questo. Un utente che vuole effettuare un pagamento include un fee nella transazione che è raccolto dal miner che genera il nuovo blocco che la include. Si introduce un altro meccanismo in questo modo: il miner è incentivato a includere prima transazioni con un fee più elevato, che quindi possono ricevere conferma più velocemente.

Modifiche di Twister alla parte Bitcoin

Twister modifica pesantemente il significato delle transazioni di Bitcoin. Il loro scopo non è più quello di registrare transazioni in ingresso e in uscita, ma quello di memorizzare coppie nome utente/chiave pubblica. In questo modo ogni utente può firmare con la propria chiave privata i messaggi, e mantenere al contempo uno pseudonimo nella rete, permettendo così l'anonimato. Il cambio di significato delle transazioni ha però portato anche ad alcune difficoltà tecniche nella realizzazione di questa tesi. In particolare il sottosistema di scripting che si occupa di verificare la validità della transazione è stato rimosso, non avendo più ingressi o uscite su cui operare. Questo ha reso molto difficile l'implementazione dell'idea di usare le multisignature di Bitcoin per aggiornare il meccanismo di moderazione in particolare.

BitTorrent

In Twister la rete BitTorrent viene usata, congiuntamente alla rete DHT, per la condivisione dei contenuti tra gli utenti. Più in generale, queste due reti sono utilizzate per memorizzare tutto quello che non dovrebbe stare nella block chain, per non farne crescere la dimensione inutilmente.

La rete BitTorrent viene usata da un utente per la pubblicazione di nuovi post. Per raggiungere questo obiettivo, è stata cambiata la semantica dell'interazione tra i peer in Twister. Mentre nell'originale rete BitTorrent l'iscrizione a un torrent permette il download di pezzi del file, nella variante di Twister l'iscrizione a un torrent equivale al "following" di un utente.

[3]É interessante notare che l'economia di Bitcoin è deflazionaria, visto il numero limitato di bitcoin generabili. Questo vuol dire che il valore del bitcoin aumenta nel tempo, ma non si pensa che questo sarà un problema vista la divisibilità del bitcoin fino a 10^{-8}

Quando un utente pubblica un nuovo messaggio, manda il messaggio have agli altri peer. Questo messaggio era originariamente pensato per segnalare il completamento da parte di un peer del download di un pezzo di file, nella rete di Twister è stato usato per segnalare la pubblicazione di un nuovo post, in modo che gli utenti iscritti possano scaricarlo. In questo schema è l'autore a fare un push dei messaggi verso i destinatari. Questa strategia inoltre funziona anche nel caso di nuovi utenti che si connettono quando il torrent è già avviato: questi si scaricano tutti i messaggi, mentre coloro che erano già iscritti scaricano solo la differenza in termini di pubblicazione.

La rete DHT viene usata per la memorizzazione di svariate informazioni, in particolare i dati profilo degli utenti. In questo senso la rete DHT di Twister assume un ruolo di dizionario distribuito, più di quanto non faccia in BitTorrent in cui svolge solo il compito di effettuare il tracking.

Modifiche di Accumunet

Accumunet porta una serie di modifiche al client Twister in un'ottica prettamente commerciale. In particolare introduce la moderazione nel sistema di Twister e cambia la semantica dei post facendoli diventare prodotti. Per moderazione si intende in senso lato la possibilità di porre un controllo ai contenuti pubblicati, in questo caso in un contesto completamente distribuito. Per arrivare a questo obbiettivo è stata impiegata la costruzione degli accumulatori crittografici.

Accumulatori crittografici

Gli accumulatori crittografici sono un costrutto studiato da Benaloh e De Mare [21] che permette di verificare l'appartenenza di un elemento in maniera efficiente. In particolare è necessario solamente l'accumulatore, l'elemento che memorizza tutti gli elementi del gruppo, e il witness, che deve essere presentato per provare l'appartenenza di un elemento al gruppo e di difficile falsificazione. L'utilizzo degli accumulatori in Accumunet è dovuto al fatto che ben si prestano all'inserimento nella block chain, dove sono naturalmente autenticati e resistenti alla modifica. Per autenticare un gruppo di utenti nella block chain la soluzione più naturale è quella di memorizzarne la lista all'interno, ma questa soluzione ha problemi di scalabilità. In particolare questa soluzione occupa uno spazio lineare nel numero di utenti, e inquina la block chain ingrandendola. Gli accumulatori invece permettono di risolvere questo problema occupando uno spazio quasi costante, dipende dalla versione utilizzata. Ci sono infatti diversi tipi di accumulatori, e quello scelto permette un'efficiente gestione dello spazio [22]:

- **Accumulatori combinatori** Questo tipo di accumulatori, descritto per la prima volta da Nyberg [23], differisce dalla costruzione RSA per l'occupazione di molto più spazio. In particolare, lo spazio occupato dall'accumulatore è proporzionale a $\lambda N \log N$, dove N è il numero di elementi accumulati e λ è il parametro che determina la difficoltà di falsificazione dell'accumulatore[4] e N è il numero di elementi accumulati. Questo significa che per accumulare 1000 elementi serve un accumulatore nell'ordine delle centinaia di kylobyte [23], supponendo di volere una probabilità di falsificazione trascurabile. La costruzione procede usando una funzione di hash che computa hash lunghissimi, più lunghi dell'accumulatore stesso di un fattore $\log N$. A questo punto gli hash vengono trasformati in stringhe di bit più corte ponendo i bit uguali a zero solo se il primo bit del blocco originale era zero, e gli hash vengono combinati tra loro tramite moltiplicazione [24]. Un pregio, del tutto teorico, di questa costruzione è che non si basa sulla correttezza del problema RSA. Sono però inadatti all'uso all'interno di Accumunet per via dello spazio occupato, in particolare all'interno della block chain.

- **Merkle Hash Tree** [25] [26] [27] Si tratta di una tipologia di accumulatori sviluppata da Jiangtao Li, Ninghui Li e Rui Xue. Per illustrarne il funzionamento verrà descritta una variante semplificata presentata in [26]. In questi accumulatori si mantiene una lista r_i di radici di un albero di Merkle, una per ogni livello di profondità dell'albero. L'accumulatore è dato dalla concatenazione di queste radici. Il witness è dato invece dal percorso di un elemento accumulato fino a una delle radici. Formalmente, dato un elemento y, il suo witness è dato da la profondità di una radice d e da elementi $e_{i,...d-1}$ tali che $h(h(...(h(h(y)|e_1)|e_2)...)|e_{d-1}) = r_d$.
  Questo accumulatore non è compatto quanto quelli di tipo RSA, occupa infatti uno spazio logaritmico nel numero di elementi. Può essere reso dinamico.

- **Accumulatori RSA** Questi accumulatori sono estremamente più efficienti in termini di spazio della loro controparte combinatoria, infatti con solamente 100 byte è possibile accumulare un numero elevatissimo di utenti. L'idea, di Benaloh e De Mare [21], è quella di usare la funzione

$$v = u^{x_1 x_2 x_3 ...} \mod n$$

[4] La probabilità di falsificare un elemento con questa costruzione è di $2^{-}\lambda$.

dove v è l'accumulatore, u è l'accumulatore vuoto, x_i è il numero primo che va accumulato e n deve essere scelto grande abbastanza. In questo caso si può osservare come sia semplice ottenere l'accumulatore presentando il witness, basta una moltiplicazione, e non conta l'ordine in cui gli elementi sono accumulati proprio perché sono all'esponente. Resta il problema di poter togliere e aggiungere elementi aggiornando allo stesso tempo i witness degli altri utenti. La soluzione adottata da Accumunet è la costruzione di accumulatore dinamico di Camenisch e Lysyanskaya [28].
L'accumulatore vuoto viene scelto come residuo quadratico del modulo, il modulo viene scelto come np, dove n e p sono numeri primi sicuri (ovvero della forma $2p-1$ con p anch'esso primo). Gli elementi da accumulare devono essere scelti come numeri primi in un particolare intervallo dipendente dai parametri dell'algoritmo. La funzione accumulatrice è la stessa di prima.
L'aggiunta di elementi all'accumulatore è molto semplice:

$$v' = v^{x'} \mod n$$

$$w' = w^{x'} \mod n$$

dove v', w', x' sono rispettivamente il nuovo accumulatore, il nuovo witness (per ogni utente) e il nuovo elemento accumulato. Purtroppo la cancellazione è molto più complessa:

$$v' = v^{x'^{-1} \mod (p-1)(q-1)} \mod n$$

$$w' = w^b v'^a \mod n$$

dove a, b sono interi tali che $ax + bx' = 1$. Non solo la cancellazione è più complessa, ma è necessaria la conoscenza dei fattori di n. Questa informazione ausiliaria permette facilmente di creare un witness per un elemento non accumulato, introducendo così un punto di attacco al sistema. La più semplice politica per la gestione di questo problema è la distruzione di questi fattori al momento della decisione di n, e la creazione di un nuovo accumulatore a ogni cancellazione. Questo sistema può essere conveniente in caso gli elementi accumulati siano pochi e le cancellazioni infrequenti. Un'altra soluzione possibile è quella di affidare il segreto a un'entità fidata, una specie di coordinatore a cui tutti i fornitori di contenuti fanno riferimento. A questo punto la sicurezza si riduce alla fiducia che si ripone in questa entità.

2.3 Altro

Nella ricerca di progetti utili allo scopo di questa tesi si è trovato almeno un'altro progetto che non rientra strettamente nelle categorie presentate precedentemente, ma ciò nonostante presenta delle caratteristiche in comune ai progetti visti e utili al raggiungimento dell'obbiettivo presentato.

2.3.1 CertCoin

CertCoin è un progetto del 2014 sviluppato in un corso del MIT da Conner Fromknecht, Dragos Velicanu e Sophia Yakoubov [27] [26]. Si tratta di un sistema decentralizzato per l'autenticazione che mira a creare una PKI (public key infrastructure) decentralizzata con bassa barriera d'ingresso. Lo scopo di una PKI è la certificazione dell'identità di un utente, in particolare garantire che la chiave pubblica sia effettivamente associata a lui. Di norma questo scopo è raggiunto firmando la chiave pubblica con la chiave privata del PKI, che però a questo punto diventa un thrusted third party. Ci sono al momento due principali sistemi PKI:

- Il certificato è firmato da un'entità centralizzata. Il sistema si basa su una chain-of-trust, e le chiavi dell'entità centrale devono essere distribuite in maniera sicura, per esempio assieme al sistema operativo. Oltre al problema della distribuzione della chiave però, esiste anche il problema che un attaccante potrebbe impadronirsi delle chiavi del PKI e forgiare così certificati falsi.

- Il certificato è firmato dagli altri utenti, nel modello della Web-of-trust PGP. Questo sistema è già decentralizzato per come è costruito, ma soffre di un'elevata barriera di ingresso, visto che per avere la firma di un altro utente l'utilizzatore sarebbe in linea di principio costretto a incontrarsi di persona con utenti partecipanti alla rete.

Il sistema è pensato come un fork di NameCoin, una moneta elettronica alternativa il cui scopo è quello di registrare nomi di dominio nella block chain (separata da quella principale di Bitcoin). NameCoin funziona registrando un dominio in cambio della moneta associata. In particolare, assieme al dominio sono registrate due chiavi pubbliche: una viene usata per firmare i messaggi provenienti dal dominio, e l'altra viene usata per firmare o revocare nuove chiavi nel caso fosse necessario.
Sono state presentate tre versioni di CertCoin, ognuna che migliora i difetti di quella precedente, sempre in senso prestazionale:

1. La prima versione non impiega nessun costrutto migliorativo e utilizza NameCoin semplicemente per memorizzare le coppie identità-chiave pubblica. Permette tutte le operazioni di un normale PKI, in particolare:

   - Registrazione di una nuova identità.
   - Aggiornamento di una chave pubblica già diffusa.
   - Ottenimento della chiave pubblica corrispondente a un'identità.
   - Verifica che una chiave pubblica corrisponda a un'identità.
   - Revoca di una chiave pubblica.

   Due sono i maggiori difetti di questa soluzione:

   - Tutti gli utenti devono conservare una copia della block chain. Considerato che attualmente la block chain di Bitcoin cresce con l'ordine di grandezza del Gigabyte al mese, se CertCoin dovesse avere anche solo una modesta diffusione questo sarebbe insostenibile per utenti mobile.
   - Le ricerche e le operazioni sono lineari nella dimensione della block chain, che deve essere scorsa tutta.

2. La seconda versione cerca di risolvere il problema dello spazio occupato dalla block chain, usando lo stesso approccio ad accumulatori già illustrato su Accumunet. In questo caso nell'accumulatore verrebbero memorizzate triple del tipo *(d, pk, exp)*, dove *d* è il dominio, *pk* è la chiave pubblica e *exp* è il periodo di validità della tripla.
   Sono stati esaminati due approcci:

   - Ogni utente mantiene il suo accumulatore. Questo approccio è infattibile con gli accumulatori standard, perchè ogni utente dovrebbe distribuire un witness a chi vuole registrare la sua identità. É stato esaminato un approccio alternativo basato su Bloom Filters, una tecnologia che permette il membership testing efficiente senza witness avendo però il difetto di produrre falsi positivi. É stato scartato perchè richiederebbe comunque troppo spazio in ambito mobile.
   - Esiste un solo accumulatore, memorizzato nella block chain. La tipologia adottata è quella del Merkle Hash Tree, quindi logaritmica nel numero di utenti. La dimensione non è un problema

sotto l'ipotesi che il numero di registrazioni sia un numero costante di volte il numero di persone sulla terra. L'aggiornamento dell'accumulatore è compito del miner che scopre un nuovo blocco.

3. La terza versione risolve il problema della ricerca efficiente di informazioni tramite l'adozione di una rete DHT di tipo Kademlia. Nella rete DHT sono memorizzate coppie *(identità, keypair)* dove *keypair* è a sua volta una coppia formata dalla chiave pubblica e dal witness dell'utente. Questo approccio presenta ancora un paio di problemi, risolvibili in maniera abbastanza rapida:

   (a) Bisogna incentivare i nodi a partecipare alla rete DHT, in modo che non sia possibile per un nodo aggiungere le proprie informazioni e poi rimuoversi lasciando agli altri peer il compito di risolvere le richieste per la sua identità. Per ovviare a questo problema, quando un peer riceve una richiesta per la chiave pubblica di un certo utente, prima di rispondere lancia un messaggio di *heartbeat* nei confronti dell'utente stesso.

   (b) La rete DHT Kademlia non è autenticata di default, e per questo vulnerabile ad attacchi di poisoning [29], routing [30] e di tipo Sybil. Per mitigare questi effetti in CertCoin, ogni utente che fa una richiesta di chiave pubblica associata ad un'identità deve fornire anche la propria di identità. Questo riduce il problema perché viene introdotto un costo implicito nella richiesta, dato dalla necessità di creare un'identità riconosciuta dalla rete.

Capitolo 3

CoMoNet

CoMoNet sta per "Content Moderated Network", ed è il fork di Accumunet sviluppato in questo lavoro che propone una soluzione ai problemi presentati nell'introduzione. Le modifiche apportate sono principalmente su due fronti, il primo è quello della creazione di un meccanismo che renda possibile aggiornare l'accumulatore in accordo a una qualche politica, il secondo è quello della creazione di una struttura per la presentazione dei prodotti degli utenti accumulati, anch'essa aggiornabile secondo criteri specifici.

3.1 Aggiornamento dell'accumulatore

Per memorizzare l'accumulatore in Accumunet si era aggiunto un campo accumulator alle transazioni. Tuttavia questo campo era utilizzato solamente in una transazione, che veniva creata convenzionalmente con il campo username valorizzato a _admin_. Una volta memorizzata nella block chain, la transazione contenente l'accumulatore poteva essere ricercata proprio tramite questo utente fittizio, il cui unico scopo è il contenere l'accumulatore. Ogni qual volta fosse necessario verificare l'appartenenza di un utente al gruppo di fornitori di contenuti, era necessario ricercare il valore nell'accumulatore nella block chain, recuperare il witness dalla rete DHT ed effettuare le necessarie operazioni. Vale la pena osservare che con questo meccanismo si possono gestire un numero arbitrario di gruppi nella rete, e che Accumunet prevedeva un sistema di caching per evitare di andare a recuperare il valore dell'accumulatore ogni volta. In Accumunet l'accumulatore veniva usato per decidere se lasciare postare o meno un utente i propri prodotti, ovvero se creare un torrent. É importante notare però che il controllo ultimo se fidarsi o meno delle informazioni che arrivano dall'esterno deve sempre essere del client che presenta le informazioni. Sarebbe dunque suo il compito di verifi-

care in ogni momento se gli utenti di cui deve presentare le informazioni sono effettivamente accumulati, e può sempre farlo visto che l'accumulatore è nella block chain condivisa da tutti. Tuttavia effettuando il controllo al momento di lasciar postare l'utente si può evitare di propagare nella rete informazioni che non dovrebbero passare, bloccando a monte tentativi di contraffazione. Nulla vieta a un eventuale attaccante di modificare il client in questo senso, ma è auspicabile che le informazioni che eventualmente venissero lanciate con un client modificato vengano soppresse il prima possibile.

In ogni caso non era stato implementato nessun meccanismo di modifica di questo accumulatore, che quindi risultava inadatto a uno scenario reale in cui i fornitori di contenuti possono entrare o uscire dal gruppo di pubblicazione per svariati motivi. Erano stati discussi tuttavia alcuni possibili scenari, dei quali il più interessante è la pubblicazione tramite quorum, che è stata qui ripresa e sviluppata.

Per l'aggiornamento dell'accumulatore l'idea principale era quella di permetterne la modifica solo a fronte di una firma di un certo numero di elementi accumulati, in modo tale che il sistema fosse quanto più democratico possibile. Il meccanismo che più si prestava a questo tipo di applicazione erano le multisignature di Bitcoin.

3.2 Multisignature

In Bitcoin una transazione viene generalmente autorizzata con una singola firma. Tuttavia Bitcoin implementa un sistema di scripting basato su stack (non Turing completo) che permette di definire sotto quali condizioni autorizzare una transazione. In particolare, il sistema supporta l'operatore OP_CHECKMULTISIG, che richiede la presenza di almeno t su n firme ECDSA per autorizzare la transazione. Per iniziare una transazione di questo tipo, viene creato un indirizzo particolare, che a differenza di quelli standard inizia con la cifra 3. Il comando usato per creare questi indirizzi è createmultisig, a cui viene passato il numero di chiavi pubbliche necessarie per sbloccare il pagamento della transazione e le chiavi stesse, in formato array json. La risposta del client rpc è un indirizzo multisignature per l'appunto e un redeem script, necessario negli step successivi[31]. Per lo sblocco dei fondi all'indirizzo multisignature è necessario creare una transazione firmata con almeno tante chiavi private quante sono le firme specificate al momento della creazione dell'indirizzo, secondo l'algoritmo ECDSA. Oltre alle firme, è necessario includere nella transazione anche il redeem script. Una volta completata la transazione, questa può essere inoltrata alla rete con il comando sendrawtransaction.

Al momento, le multisignature sono utilizzate con un valore di $n \leq 3$, tuttavia il limite assoluto è $n = 20$. Il limite di $n \leq 3$ è al momento oltrepassabile solo usando il *pay-to-scrypt-hash*, ovvero includendo l'hash dello script di autorizzazione al posto dello script stesso nella transazione. Anche in questo caso tuttavia non si può andare oltre a $n = 15$.

Benché questa tecnologia rispondesse pienamente ai requisiti di aggiornamento dell'accumulatore, si è scelto di non farne uso per due principali motivi:

- Difficoltà di implementazione: il client Twister originale ha rimosso il supporto allo scripting nelle transazioni, supporto necessario all'implementazione delle multisignatures. Nonostante pezzi di codice dell'originale client Bitcoin compatibili con le multisignatures siano rimasti, Twister non fa uso di questa caratteristica e la sua aggiunta si rende complicata dal fatto che Twister non opera su transazioni finanziarie. Questo significa che una transazione nella block chain non è indirizzata a qualcuno o qualcosa in particolare, ma contiene solo il nome utente e la corrispondente chiave pubblica. L'assenza di un indirizzo a cui mandare la transazione è un problema bloccante: per effettuare una transazione multisignature in Bitcoin, il primo passo è la generazione di un indirizzo di questo tipo, che contiene le informazioni di quali chiavi devono firmare per autorizzare la transazione (insieme allo hash script). Si può ipotizzare di usare un sistema simile in Accumunet usando l'indirizzo multisignature al posto del campo username. Questo rende però difficoltoso il recupero dell'accumulatore: un client non avrebbe modo di sapere a priori quali sarebbero questi indirizzi che contengono l'accumulatore.

- Flessibilità del sistema: al momento le multisignatures di Bitcoin sulla rete ufficiale sono estremamente limitate [32], permettendo in linea teorica un numero massimo di firme richieste pari a 20, e in pratica ponendo un limite molto più stretto, con le tipologie più diffuse di transazioni multisig che si attestano sulle 2-of-3. Anche se si utilizzassero tutte le firme disponibili, ovvero 20, e anche se si rilassasse il limite di dimensione del P2SH redeem script, resterebbe il problema che il sistema richiederebbe di fissare un numero massimo all'inizio di partecipanti, e rende comunque non banale l'implementazione di una politica di aggiornamento dipendente dal numero di partecipanti (per esempio nel caso si volesse rendere necessaria la firma di almeno metà degli elementi per la modifica).

3.2.1 Implementazione alternativa

Per aggirare questi problemi si è scelto di adottare una strategia ispirata alle multisignatures ma di diversa natura. Per iniziare, si è previsto un meccanismo di recupero dell'accumulatore basato sul campo username. L'accumulatore i-esimo è pubblicato con il campo username "_admin_i". Quando è necessario recuperare l'ultimo accumulatore valido, un client parte dalla prima transazione e le recupera tutte fino a quando la ricerca nella block chain fallisce, segnalando dunque l'assenza di altri accumulatori. Questo sotto ipotesi che l'entità che pubblica aggiornamenti, i fornitori di contenuti stessi riuniti presumibilmente, crei una catena lineare senza interruzioni. In quest'ultimo caso si potrebbe implementare una sorta di look-ahead di un certo numero di elementi per avere la sicurezza di non perdere nuove transazioni. Il codice che effettua questo controllo è nel listato 3.1.

```
1  // Recupera la prima transazione
2  string next_try;
3  next_try = admin_str + itostr(1);
4  printf( YELLOW "\n%s\n", next_try.c_str());
5  if (!GetTransaction(next_try, txAccumulator, acc_hashBlock))
       {
6    printf( RED "\nCan't retrieve the first accumulator, aborti
       ng\n" RESET "\n");
7    return ACC_ERROR;
8  }
9
10 // 1024 tentativi (arbitrario)
11 for(int i = 2; i < 1024; i++) {
12   next_try = admin_str + itostr(i);
13
14   CTransaction temp;
15   uint256 temp2;
16
17   // Se non lo trova \'{e} arrivato all'ultimo, interrompi.
18   if (!GetTransaction(next_try, temp, temp2)) {
19     printf( YELLOW "\nAdmin transaction %s not found\n", next
       _try.c_str());
20     break;
21   } else {
22     previousAccumulator = txAccumulator;
23     txAccumulator = temp;
24   }
25 }
26
```

Listing 3.1: Ricerca dell'ultimo accumulatore nella block chain.

Una volta trovato l'ultimo accumulatore valido, un client recupera dal campo pubKey un indirizzo DHT dal quale prelevare una lista di firme dell'accumulatore corrispondente. Si è scelto di memorizzare questa sorta di puntatore alla lista anziché la lista stessa per una considerazione di spazio occupato in block chain: visto che quest'ultima è condivisa da tutti i partecipanti della rete, si vuole evitare di piazzare una grande quantità di dati al suo interno. Nel caso l'indirizzo DHT non sia raggiungibile, si può pensare a una soluzione ridondata in cui il campo pubKey contiene più indirizzi, ed eventualmente un indirizzo di default di backup condiviso da tutti i client a cui chiedere la lista. Una volta recuperata la lista, il client utilizza lo stesso il meccanismo di verifica di una signature interno a Bitcoin, che permette di verificare una firma senza la chiave pubblica dell'utente, a patto di avere anche il contenuto firmato, ovvero l'accumulatore in questo caso. Il codice per questo procedimento si può vedere nel listato 3.2. La verifica ha successo solo se c'è almeno la metà più uno delle firme di coloro che hanno firmato l'accumulatore precedente. Per sapere quanti hanno firmato l'accumulatore precedente un client può semplicemente tenere traccia delle transazioni mentre scorre la block chain per recuperare l'ultimo accumulatore. Il caso base, cioè il primo accumulatore, è gestito a parte e richiede che nella lista le firme siano tutte valide.

```
1  BOOST_FOREACH(const Pair& signature_pair, signatures) {
2    string username = signature_pair.name_;
3    string signature = signature_pair.value_.get_str();
4
5    printf( YELLOW "\nSignature for: %s\n", username.c_str());
6    printf( YELLOW "\nSignature: %s\n", signature.c_str());
7
8    Array temp;
9    temp.push_back(username);
10   temp.push_back(signature);
11   string uppercase_acc = boost::to_upper_copy(txAccumulator.a
       ccumulator.ToString());
12   temp.push_back(uppercase_acc);
13
14   // Qui avviene l'effettiva validazione
15   Value validate = verifymessage(temp, false);
16   bool valid = validate.get_bool();
17   printf( YELLOW "\nValid: %d\n", valid);
18
19   if(valid) {
20     validated_signatures++;
21   }
22 }
```

Listing 3.2: Verifica del numero di firme valide.

La pubblicazione di un nuovo accumulatore viene fatta in due passaggi:

1. Esternamente alla rete, i partecipanti si riuniscono a firmare il nuovo accumulatore. Ognuno produce una firma dell'accumulatore mediante il comando createsignature ereditato dal client originale Satoshi. La lista delle firme viene quindi pubblicata sulla rete DHT attraverso il comando dhtput, secondo la sintassi seguita nel listato 3.3.

```
1 ./twisterd -genproclimit=1 -rpcuser=user -rpcpassword=pwd
2 -rpcallowip=127.0.0.1 -rpcport=48001 dhtput utente1
3 signature s '\{"utente1":"IERBSmoEVSPPX0qAWwAfQCiOCbnbZX+byDnVAgx9+RtUxlZXiXykPkmRu7L9XQtt576uMbbgxn7xc5A+3k4ZSno=","utente2":"IIosg4mvqyeFMuzxgcGnP1tKlo2YQxldrsa+lF85dXdMbU4ix8weCky/UlVZnajPf4NgiQBqXXGyt4knWPyUBQM="\}' utente1 0
4
```

Listing 3.3: Inserimento della lista di firme in DHT

Viene poi creata una transazione da inserire nella rete mediante il comando createrawaccumulatortransaction, con la sintassi del listato 3.4.

```
1 ./twisterd -genproclimit=1 -rpcuser=user -rpcpassword=pwd
2 -rpcallowip=127.0.0.1 -rpcport=48002 createrawaccumulatortransaction _admin_2 ffffff 07EEDC174BA9061088BC764189ECB85D351A615F0CA5781C77436735F23224C5 utente1 010000000009085f61646d696e5f3208077574656e746531212007eedc174ba9061088bc764189ecb85d351a615f0ca5781c77436735f23224c5b6500000
3
```

Listing 3.4: Creazione della transazione contenente l'accumulatore.

2. La transazione in formato raw viene inserita da un client nella block chain (possibilmente dopo aver verificato che le firme siano valide, anche se può comunque esistere un client malevolo).

```
1 ./twisterd -genproclimit=1 -rpcuser=user -rpcpassword=pwd
2 -rpcallowip=127.0.0.1 -rpcport=48002 sendrawtransaction
    010000000009085f61646d696e5f3108077574656
    e7465312120450da364ae10b42c83f180d01fecf5cbd0901d4b1
    b8eed22d8490d46a42a65e74b100000
3
```

Listing 3.5: Inserimento della transazione nella block chain.

3.3 Gestione della struttura del portale

Con struttura del portale si intende una serie di parametri che determinano la composizione e l'aspetto della pagina di presentazione dei contenuti, in termini di ordine riservato ai contenuti, spazio dedicato e altri parametri. Questa struttura è attualmente visualizzata dagli stessi client che mantengono la block chain, indirizzando un browser in locale, ma non è escluso che i client possano a loro volta funzionare da server e distribuire la struttura a client più leggeri, che non mantengono la block chain (in ambito mobile per esempio).

Per la memorizzazione della struttura si sono adottate le stesse scelte impiegate per la memorizzazione dell'accumulatore. Quindi la struttura è memorizzata in una catena di transazioni, dove l'i-esima transazione ha il campo username valorizzato a _structure_i. Un client scorre la catena analogamente a quanto succede per l'accumulatore.

L'unica differenza sostanziale è il meccanismo di approvazione e firma della struttura: è sufficiente che contenga la firma di almeno metà più uno dei partecipanti che hanno firmato l'accumulatore corrente.

L'intera struttura è memorizzata in una transazione della block chain. Si tratta di una soluzione accettabile vista l'esigua quantità di parametri utilizzati. Nel caso però di strutture più complesse, magari ispirate dalle strutture usate da ZeroNet, è ipotizzabile una soluzione ibrida, che possa contenere parte dei parametri, magari quelli più importanti, direttamente nella transazione e gli altri nella DHT. Questo può chiaramente causare problemi in un'applicazione pratica reale nel caso l'indirizzo DHT non sia raggiungibile, quindi bisogna adottare una soluzione di replicazione per tutelarsi da questo tipo di evento. La sintassi adottata è una json molto semplice, come si può vedere nel listato 3.6. Si può pensare di complicare a piacimento questo schema aggiungendo parametri di struttura e di stile, ma in questo caso ci si è limitati a un'impostazione tabellare di cui viene specificato il numero di colonne, di righe e l'ordine in cui devono comparire gli utenti accumulati.

```
1 ./twisterd -genproclimit=1 -rpcuser=user -rpcpassword=pwd -rp
    callowip=127.0.0.1 -rpcport=48002 createrawstructuretransa
    ction _structure_1 utente1 '{"rows":3,"columns":3,"order"
    :["utente1","utente2"]}' null
2
3  01000000000d0c5f737472756374757265 5f3108077574656
    e74653135347b22726f7773223a332c22636f6c756d6e73223a332c226
    f72646572223a5b227574656e746531222c227574656e746532225d7de
    d4b0000
4
```

Listing 3.6: Creazione della transazione contenente la struttura.

3.4 Visualizzazione della struttura

La visualizzazione dei contenuti avviene tramite un browser. Al momento, la parte html di visualizzazione dei prodotti e il client derivato da Bitcoin/Twister sono distribuiti insieme, sia in Accumunet sia in CoMoNet, ma non è esclusa una separazione tra i due motivata per esempio da una volontà di proporre una versione mobile del sistema. Per accedere alla home page, è sufficiente collegarsi all'indirizzo locale del client CoMoNet mentre è in esecuzione. Il web server interno provvede a fornire la pagina di ingresso, che esegue del codice javascript per fare una chiamata RPC al client stesso. La chiamata recupera per prima cosa la transazione della struttura. All'interno di questa transazione è presente la lista dei fornitori di contenuti di cui recuperare i prodotti e l'ordine in cui mostrarli.
Supponendo invece l'adozione di un approccio ibrido, una volta recuperata la transazione il browser dovrebbe mandare un'altra richiesta alla DHT per il recupero dei dati mancanti.
Una volta recuperato tutto il necessario a visualizzare la struttura, si può procedere al recupero dei dati effettivi da presentare sulla pagina. La tecnologia usata in questo caso è la libreria AngularJS, che si occupa delle chiamate asincrone per il recupero dei dati dal client Accumunet e ne visualizza i risultati tramite data-binding con gli elementi della pagina. Vale la pena osservare che un meccanismo di cache, benché non sia stato implementato in questa versione, può aiutare a ridurre le latenze specialmente nel caso il sistema dovesse essere utilizzato da più utenti su Internet, dove la qualità della connessione è fortemente variabile. Bisognerebbe tuttavia pianificare attentamente il time-to-live del contenuto in cache, per evitare problemi di obsolescenza dei contenuti.

Capitolo 4

Possibili sviluppi

Il software CoMoNet è poco più di un proof-of-concept, e qui si andranno ad analizzare le possibili estensioni a questo sistema. In particolare verranno analizzati una strategia alternativa di aggiornamento dell'accumulatore, quali incentivi si potrebbero dare agli utenti per partecipare alla rete e come la rete potrebbe espandersi su mobile e auto-sostenersi in caso di carichi elevati.

In ultima analisi verrà presentata una panoramica del web distribuito, una direzione che prevede di decentralizzare il protocollo HTTP di cui il social network Twister e anche il suo derivato Accumunet è un esponente [1].

4.1 Threshold signatures

Le threshold signatures [33] sono un'alternativa alle multisignatures di Bitcoin. In sintesi si tratta di un sistema che permette più flessibilità rispetto alle multisignatures e per questo adatto all'uso in Accumunet, ma che ha diversi trade-off, primo tra tutti l'elevata complessità.
Un algoritmo di firma a soglia fa esattamente quello che fanno le transazioni multisignatures, ovvero permette la firma solamente con l'autorizzazione di almeno t tra n entità.
La differenza sostanziale tra l'algoritmo presentato e le multisignatures sta nel fatto che non ci sono limiti a t o a n di tipo tecnico, se non quelli dettati dalla complessità computazionale. Inoltre il prodotto finale è un'unica firma, a differenza delle multisignatures dove è necessario il sottosistema di scripting di Bitcoin per estrarre dallo stack tutte le chiavi pubbliche che devono essere usate per firmare, una per una.
Un'altra importante caratteristica dell'algoritmo in esame è che non rivela più informazione di quanto sia necessaria. Si prenda per esempio il classico schema a soglia di Shamir [34]. In questo schema un segreto che si suppone

debba essere condiviso da n utenti e recuperabile da t, viene codificato come il termine costante di un polinomio. A questo punto a ogni utente viene distribuita una coppia (x, f(x)) dove f è il polinomio, e il segreto può essere ricostruito con i coefficienti delle t entità. Si potrebbe pensare di conservare in questo modo la chiave privata per la firma, nel caso di Accumunet per la firma dell'accumulatore, ma il problema è che una volta riuniti i coefficienti la chiave privata è ricostruita interamente e chi li ha riuniti ne viene a conoscenza. Nell'algoritmo di firma a soglia discusso qui invece le entità possono firmare ripetutamente qualsivoglia oggetto senza dover per questo mai svelare la chiave privata.

4.1.1 Struttura dell'algoritmo

L'algoritmo è molto complesso e si prova qui a riassumerne i tratti salienti. Per iniziare gli autori del paper presentano una notazione con cui descrivere sia l'algoritmo DSA sia l'algoritmo ECDSA. Mentre il primo è basato sullo schema di firma ElGamal [35], il secondo è basato sulla crittografia a curva ellittica [36].

- Si parte prendendo un gruppo ciclico $\mathcal{G}$ di ordine q, generato dall'elemento G. Si definisce poi una funzione H da $\mathcal{G}$ a Z_q.
- Si genera una chiave privata x prendendo un elemento a caso da Z_q, e una corrispondente chiave pubblica $y = G^x$.
- Si calcola l'hash del messaggio M da firmare come $m = H(M)$, e si sceglie k come un elemento a caso di Z_q.
- La firma è data dalla coppia (r, s) dove $r = H(G^k)$ e $s = k^{-1}(m + xr) \mod q$.
- Per verificare la firma, prima si controlla che $r, s \in Z_q$. Poi si calcola $R = G^{ms^{-1} \mod q} y^{rs^{-1} \mod q}$ e se $H(R) = r$ la firma è verificata.

Specializzando gli elementi al punto precedente si possono ottenere i due schemi di firma:

- **DSA** Si scelgono q, p come numeri primi molto grandi e tali che q divida $p - 1$. Il gruppo $\mathcal{G}$ è Z_q^*, ovvero l'insieme di tutti gli elementi invertibili di Z_q. La moltiplicazione è fatta modulo q e la funzione di hash è $H(R) = R \mod q$.

- **ECDSA** Si usa una curva ellittica di cardinalità q come $\mathcal{G}$, la moltiplicazione è l'operazione di gruppo sulla curva e la funzione di hash è $H(R) = H_x \mod q$ dove H_x è la coordinata x del punto R.

Lo schema estende una strategia di MacKenzie e Reiter del 2001 [37]. Questa strategia è limitata al caso di due entità, e sfrutta uno schema di crittografia omomorfica. L'idea è che uno dei due ha la chiave segreta di questo schema omomorfico, e la usa per cifrare la sua parte della firma. Questa viene quindi inviata al secondo, che firma con la sua parte di chiave privata completando la firma ma senza decrittare la parte ricevuta. Lo scambio è accompagnato da una prova a conoscenza zero, per garantire che ogni parte partecipi al protocollo.
Passando allo schema qui in esame, l'estensione lavora in maniera simile ma con un numero variabile di partecipanti. La modifica più importante è che ora nessuno è in possesso della chiave segreta per decifrare la firma come succedeva nel caso semplificato di prima, ma la chiave è distribuita tra tutti i partecipanti. Lo schema di crittografica omomorfica utilizzato è quello di Paillier [38].
L'algoritmo, supponendo che partecipino n entità, richiede $3n - 2$ round per essere completato. Senza riportare per intero il procedimento visibile in figura 4.1, nel primo round ogni entità genera un proprio segreto la crittografia omomorfica e passa il messaggio e la codifica di questo segreto secondo Pallier alla prossima entità. All'ultima entità arrivano tutti i segreti accumulati nel corso del procedimento e può generare la quantità R descritta in precedenza. Nel secondo round tutti generano la quantità R a ritroso, e nel terzo viene generata una prova a conoscenza zero per ogni partecipante. All'ultimo passaggio viene generata una quantità μ con i dati raccolti che viene poi distribuita a tutti i partecipanti, e quindi tutti possono decifrarla con l'algoritmo di Pallier per ottenere la coppia (r, s).
É importante notare che questo algoritmo realizza una firma a soglia di tipo n su n: per realizzare una politica di tipo t tra n gli autori consigliano di usare uno schema combinatorio che consideri tutte le possibili $\binom{n}{t}$ combinazioni.

4.1.2 Comparativa rispetto a un approccio multisignature

Il vantaggio principale delle threshold signatures rispetto alle multisignatures è la flessibilità. In questa tesi si è dovuto adottare un approccio alternativo alle multisignatures proprio per il loro limite tecnico di non poter scalare a un numero arbitrario di utilizzatori. Con le threshold signatures invece è possibile coinvolgere un numero arbitrario di partecipanti, anche se non

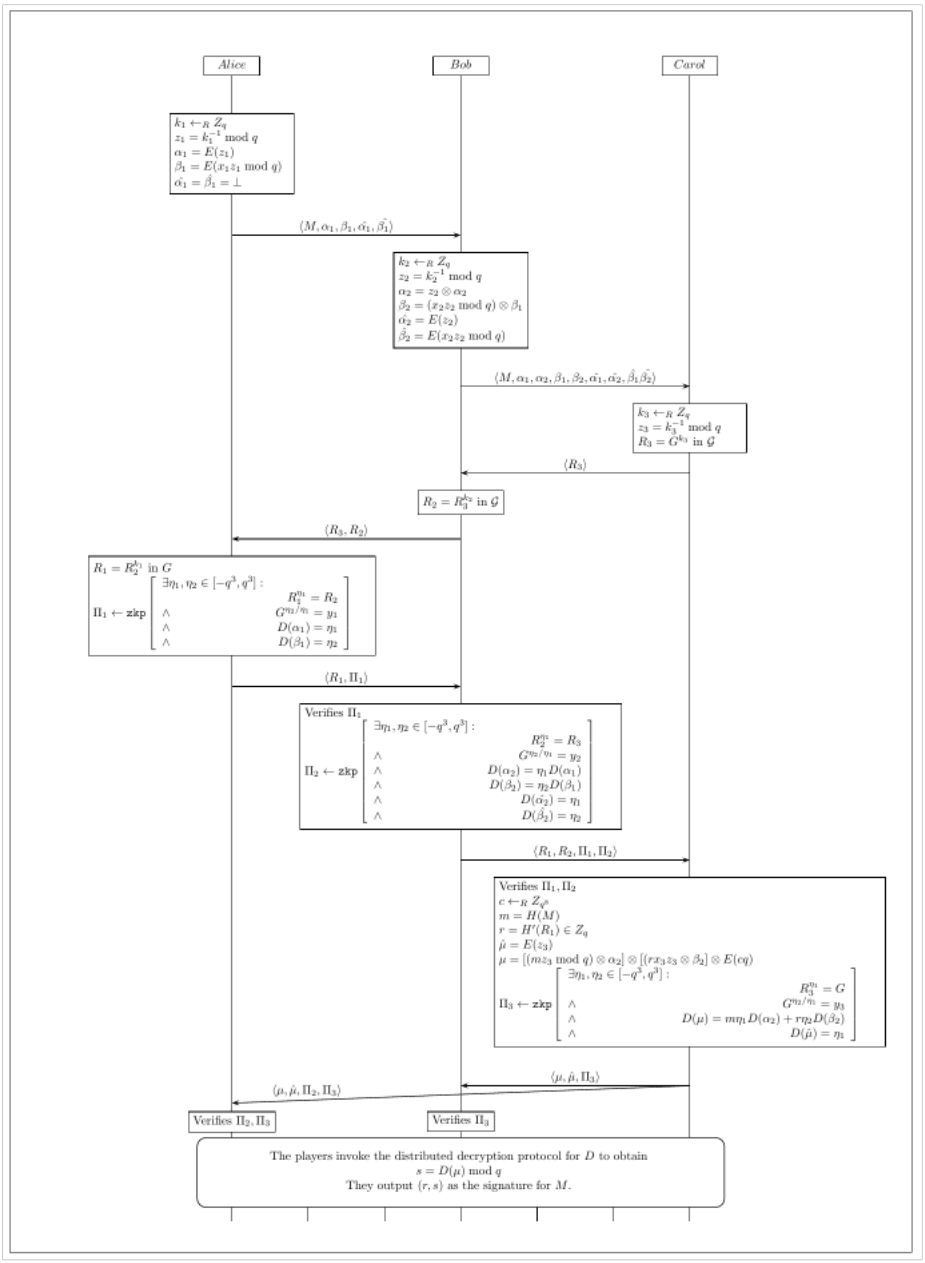

Figura 4.1: Schema del protocollo per un 3-3 ([33])

elevatissimo, vista l'elevata complessità computazionale dell'algoritmo, che richiede diverse prove a conoscenza zero, e vista la natura interattiva della generazione delle firme. Per contro, l'approccio adottato in questa tesi permette una generazione offline delle firme, che possono poi essere pubblicate in un secondo momento, e il trade-off in tempo computazionale si ha nello spazio occupato nella rete DHT o block chain, a seconda di dove si decida di mettere le firme. In questo caso si è ritenuto opportuno adottare l'approccio più semplice.

4.2 Incentivi all'utilizzo

CoMoNet allo stato attuale è pensata per l'utilizzo solo da parte dei fornitori dei contenuti. In realtà i consumatori avrebbero ben poco stimolo a installarsi il client solamente per visualizzare i contenuti, tanto più che facendolo entrerebbero a far parte della rete condividendo banda e potenza di calcolo per mantenere la block chain.
Ispirandosi quindi direttamente alla soluzione trovata da Miguel Freitas per il social network Twister [8], ovvero quella di cambiare il significato della transazione coinbase, si può creare un incentivo alla partecipazione. In particolare, mentre in Twister la transazione coinbase generava un messaggio pubblicitario scritto da chi scopriva un nuovo blocco, in CoMoNet si può pensare di usare la stessa transazione in 3 modi:

- Per generare un messaggio pubblicitario come nell'originale Twister. Il problema di questo approccio è che nessuno vieta di creare un client che ignora questi messaggi.

- Per indicare quale fornitore assume una posizione privilegiata all'interno della home page iniziale. Supponendo ad esempio che la home page sia strutturata a lista, lo scopritore di un nuovo blocco andrebbe a occupare la prima posizione fino alla scoperta del blocco successivo. Anche qui, alla fine, è delegato al client il rispetto di questa disposizione.

- Il terzo modo è quello che si propone come soluzione al problema della partecipazione degli utenti. Un utente che dovesse scoprire un nuovo blocco avrebbe diritto a un premio di qualche tipo. Nel caso che i fornitori di contenuti fossero venditori, questo premio potrebbe assumere la forma di sconto o promozione. Come far rispettare questo premio ai fornitori è un problema diverso. Il fornitore è libero di ignorare questo premio, ma a questo scopo si potrebbe ideare un meccanismo ispirato ai contratti riccardiani di OpenBazaar. L'idea è che lato fornitore il

contratto è forzato e implicito. Il consumatore pubblica il suo diritto allo premio, designando al contempo un arbitro. A questo punto il fornitore può rimborsare il valore dello sconto in Bitcoin. Il problema è che a differenza dei contratti riccardiani il fornitore non ha impegnato nulla all'inizio, quindi questo schema così com'è non risolverebbe nulla. Si può però ipotizzare che ogni fornitore all'inizio generi dei contratti appositi a questo scopo, che assumono tra l'altro un ruolo simile al *proof-to-miner*. Fondamentale in questo schema diventa l'arbitro.

4.3 User friendliness e supporto mobile

Ora come ora CoMoNet è un software poco user-friendly. In realtà, per l'obiettivo preposto, è plausibile che un fornitore di contenuti possa apprendere come installare e utilizzare CoMoNet nello stato attuale, o con alcuni miglioramenti, per proporre i suoi contenuti. Quello che non è plausibile è che un utente prenda parte alla rete solo per prendere visione dei contenuti. Per quanto riguarda gli stimoli che potrebbe avere un utente a partecipare al mantenimento della block chain, si veda il paragrafo precedente. Qui invece si proporrà una soluzione con cui gli utenti possano visualizzare i contenuti senza per questo farsi carico dell'onere di una block chain.
L'idea qui proposta è quella di disaccoppiare la rete dei fornitori di contenuti dalla rete dei consumatori, fornendo a quest'ultimi un client semplificato, che si interfaccia alla rete CoMoNet per recuperare le informazioni. Qui tornano utili le idee presentate inizialmente nella sezione CDN distribuite. In particolare si potrebbe adottare una soluzione ispirata all'approccio PAC. In questo caso i fornitori assumono il ruolo di CDN, pur essendo una rete P2P loro stessi, e i consumatori possono essere dotati di un'applicazione, sia essa nativa o web, che permetta loro di accedere ai contenuti. Alcuni dei nodi di CoMoNet assumerebbero il ruolo di "server", o comunque di punto di accesso, ed estrarrebbero le informazioni dalla block chain e dalla DHT per poi mandarle al client. Sorge qui il problema della sicurezza della comunicazione tra client e nodo CoMoNet, e anche di fiducia tra i due. Una possibile soluzione a questo problema potrebbe essere il rilascio dell'applicazione tramite un canale sicuro, come potrebbe essere un'estensione del browser, e inserire al suo interno dei nodi fidati di bootstrap che devono essere fidati e raggiungibili quasi sempre.
Dal punto di vista prestazionale si può fare caching delle informazioni presso il nodo di ingresso di CoMoNet.
Questa soluzione permette facilmente di creare un client mobile per accedere ai contenuti, ed è probabilmente anche l'unica plausibile vista l'improponibi-

lità di eseguire il client CoMoNet su un dispositivo mobile, specialmente per la presenza della block chain.

4.4 Web distribuito

Il concetto di web distribuito è un'idea che prevede la sostituzione dell'attuale meccanismo di distribuzione dei contenuti, basato su un modello client-server e guidato dal protocollo HTTP, con una strategia più decentralizzata. Chiaramente questa sostituzione non può essere effettuata a breve termine, nè è probabile che le soluzioni presentate abbiano successo vista l'enorme diffusione del protocollo menzionato e vista l'incredibile quantità di know-how sviluppato nel corso degli anni. Nonostante ciò, vi sono buone ragioni per studiare comunque sistemi decentralizzati, sia per i fornitori sia per i fruitori di servizi:

- Miglioramento in termini di scalabilità e affidabilità. In questo momento si utilizzano le CDN per questi obbiettivi, ma le CDN stesse stanno esplorando una soluzione più distribuita come si è illustrato in questo lavoro. L'idea fondamentale è che eliminando il single point of failure i parametri menzionati migliorano.

- Misura contro la censura, in quanto una disponibilità distribuita in posti geograficamente diversi può ostacolare un'azione di oscuramento da parte delle autorità.

- Migliore protezione della privacy, in quanto creatore e fornitore del contenuto vengono disaccoppiati e il creatore non può facilmente ricreare statistiche di accesso alla risorsa.

Purtroppo non ci sono solo vantaggi in questo approccio. Lo svantaggio principale è che alcune applicazioni web fanno uso dell'attuale centralizzazione. In particolare è un problema il mantenimento della sessione, in quanto una risorsa può essere richiesta prima da un peer e poi da un altro, ed è un problema il fatto che il creatore del contenuto può voler eseguire computazioni sul server. Quest'ultimo problema è mitigato in parte dalla crescente potenza di calcolo delle tecnologie di scripting lato client, ciò non toglie che il calcolo lato server può essere usato anche per questioni di sicurezza, riservatezza del codice e non solo per questioni di prestazioni.
Ci sono diverse tecnologie rivolte a questo tipo di strategia: lo stesso Twister, IPFS, Tahoe-LAFS, Storj, Maidsafe, FileCoin sono solo alcuni esempi, che si concentrano principalmente sulla memorizzazione dei dati in maniera

distribuita. Ci sono poi alcuni progetti che tentano la condivisione di file utilizzando la tecnologia WebRTC, come Peerjs, peerCDN and Peer5 [1]. É interessate esaminare almeno uno di questi casi da vicino, per capire quali sono i problemi del web attualmente e quali soluzioni sono proposte.
Si prenda per esempio IPFS[1] [39]. Si tratta di un progetto molto giovane, nato nel 2014, in cui si cerca di combinare un gran numero di tecnologie distribuite come git, BitTorrent, Kademlia e Bitcoin. Il progetto è piena evoluzione, quindi alcuni dettagli implementativi e la forma precisa del prodotto finale sono ancora in fase di determinazione, ma nonostante ciò si possono delineare le caratteristiche comuni ai progetti di questo tipo, CoMoNet incluso.
Questo sistema di storage distribuito è basato su uno stack di componenti:

1. **Identities** Gestisce l'identità dei nodi, in particolare un nodo è rappresentato dall'hash di una chiave pubblica. Qui si può osservare l'evidente parallelismo con Bitcoin, Twister e lo stesso CoMoNet.

2. **Network** Gestisce la comunicazione tra nodi. La cosa più interessane di questo strato è che non necessita per forza del protocollo TCP/IP, è sufficiente che un rimpiazzo rispetti l'interfaccia usata. Difficile immaginare comunque una sostituzione del protocollo a questo livello per l'immensa diffusione del TCP/IP.

3. **Routing** Per l'instradamento del traffico tra nodi viene impiegata una variante della rete DHT ispirata alla Coral DSHT. É comunque previsto che l'implementazione del routing possa essere sostituita, come allo strato precedente. Anche qui si può notare il parallelismo con software come Twister e CoMoNet, che usano anche loro una DHT.

4. **Exchange** Lo scambio di file avviene con l'uso di un protocollo ispirato a BitTorrent e denominato BitSwap. L'idea alla base è sempre quella di mantenere una lista di pezzi di file posseduti e una di pezzi da recuperare. Le modifiche fondamentali sono due: si tratta di un protocollo più flessibile di BitTorrent, senza limitarsi a usare solo i contenuti di un file .torrent, e viene mantenuto un registro degli scambi da parte dei peer come misura per contrastare il leeching e incentivare la condivisione. Qui è palese l'ispirazione alla tecnologia della block chain, benché questo registro non sia eterno ma soggetto a cancellazioni in base alla necessità. Anche l'idea di usare una variante del BitTorrent è palesemente ispirato a Twister.

[1] InterPlanetary file system.

5. **Objects** Per l'indirizzo dei file il sistema usa l'hash crittografico del contenuto. In realtà l'immagazzinamento avviene con un Merkle DAG, struttura dati così chiamata perchè generalizzazione di un Merkle Tree. Gli indirizzi sono della forma /ipfs/<hash-of-object>/<name-path-to-object>.

6. **Files** A un più alto livello i file sono versionati con un sistema molto simile al git.

7. **Naming** L'uso di hash crittografici è molto scomodo per gli umani, quindi per alleviare il problema si pensa di fare ricorso al già collaudato meccanismo DNS, servizi di accorciamento degli URL, o utilizzando NameCoin per memorizzare nomi facilmente ricordabili.

Com'è possibile notare in alcuni punti il progetto rimane vago, segnale del fatto che non sono ancora stati fatti sufficienti studi in merito. É evidente però la somiglianza di questo progetto con l'oggetto di questa tesi. La forma finale del prodotto è ancora incerta, ma le ipotesi principali puntano a farlo diventare prima un file system in piena regola che possa essere montato da sistemi unix, poi eventualmente farlo diventare una CDN e nella migliore delle ipotesi potrebbe sostituire il web.
Una possibile affermazione di queste tecnologie sembra essere molto remoto. Sarebbe necessaria la concomitanza di tre fattori: sinergia tra le tecnologie coinvolte, un caso d'uso pratico e una campagna di marketing/passaparola virale. Ma l'interazione tra le tecnologie è in piena fase di studio, un caso di utilizzo che porti la massa a utilizzare un sistema del genere non sembra essere vicino, senza contare la pessima usabilità di questi sistemi attualmente, e una campagna di pubblicità e/o il passaparola non hanno senso senza almeno uno dei due fattori precedenti.
Tuttavia lo studio di queste tecnologie è giustificato da una crescente necessità di decentralizzazione, visto che il web sta diventando sempre più accentrato attorno a pochi servizi. La speranza è che uno sviluppo tecnologico sufficiente possa sbloccare anche gli altri due fattori menzionati.

Capitolo 5

Test funzionali

Il software sviluppato per questa tesi è stato testato in due ambienti, in locale e su una rete di 3 macchine virtuali, la cui configurazione è riportata in appendice. Per il test in locale lo script prevede l'avvio di 3 client tutti collegati all'indirizzo 127.0.0.1, ma su 3 porte diverse. La demo può essere avviata eseguendo lo script 5.1, che è una versione modificata di uno script usato originariamente per la prima versione di CoMoNet.

```
1  #!/bin/bash
2
3  function coloredEcho(){
4    local exp=$1;
5    local color=$2;
6    if ! [[ $color =~ '^[0-9]$' ]] ; then
7      case $(echo $color | tr '[:upper:]' '[:lower:]') in
8      black) color=0 ;;
9      red) color=1 ;;
10     green) color=2 ;;
11     yellow) color=3 ;;
12     blue) color=4 ;;
13     magenta) color=5 ;;
14     cyan) color=6 ;;
15     white|*) color=7 ;; # white or invalid color
16     esac
17   fi
18   tput setaf $color;
19   echo -e $exp;
20   tput sgr0;
21 }
22
23 BIN=./aLaunch
24
25 echo "Do you want to restart all? [y/N]"
26 read -n1 answer
```

```
27 if [ "$answer" == "y" ]
28 then
29   $BIN delete 1 3
30 else
31   if [ "$answer" != "n" ]
32   then
33   echo -e "\nAssuming NO\n\n\n"
34   fi
35 fi
36
37 $BIN lboot 1 3
38 $BIN rhtml 1 3
39 $BIN start 1 3
40 sleep 5
41 $BIN connect 1 2
42 $BIN connect 2 1
43 $BIN connect 3 2
44 $BIN cmd 1 addwitnesstouser utente1 357d626250e41eb0390d60e4
       c33859369681fec371ab532e428b63059e97c6a7
45 $BIN cmd 2 addwitnesstouser utente1 357d626250e41eb0390d60e4
       c33859369681fec371ab532e428b63059e97c6a7
46 $BIN cmd 3 addwitnesstouser utente1 357d626250e41eb0390d60e4
       c33859369681fec371ab532e428b63059e97c6a7
47 sleep 0.5
48 $BIN cmd 2 addwitnesstouser utente2 26ff76d0f66b8403bda6534fc
       e3dbcaf47ffd5eb7e06720dd3d58aee587a39a5
49 $BIN cmd 1 addwitnesstouser utente2 26ff76d0f66b8403bda6534fc
       e3dbcaf47ffd5eb7e06720dd3d58aee587a39a5
50 $BIN cmd 3 addwitnesstouser utente2 26ff76d0f66b8403bda6534fc
       e3dbcaf47ffd5eb7e06720dd3d58aee587a39a5
51 sleep 0.5
52 $BIN cmd 2 addwitnesstouser utente3 450da364ae10b42c83f180d01
       fecf5cbd0901d4b1b8eed22d8490d46a42a65e7
53 $BIN cmd 1 addwitnesstouser utente3 450da364ae10b42c83f180d01
       fecf5cbd0901d4b1b8eed22d8490d46a42a65e7
54 $BIN cmd 3 addwitnesstouser utente3 450da364ae10b42c83f180d01
       fecf5cbd0901d4b1b8eed22d8490d46a42a65e7
55 sleep 0.5
56 coloredEcho "starting miner..." yellow
57 $BIN cmd 1 setgenerate true 1
58 sleep 0.5
59 coloredEcho "Creating structure" yellow
60 # Transazione di struttura
61 $BIN cmd 1 sendrawtransaction 01000000000d0c5
       f737472756374726 55f3108077574656e74653129287b22726
       f7773223a332c22636f6c756d6e73223a332c226f72646572223
       a227574656e746531227dc59b0000
62 coloredEcho "\nwait a moment for the dht to be ready" red
63 sleep 5
```

```
64 coloredEcho "\nPress any button when the dht is loaded" blue
65 read -n1
66 coloredEcho "Publishing order for structure" yellow
67 $BIN cmd 1 dhtput utente1 order s '["utente1","utente2"]' ute
       nte1 0
68 coloredEcho "Publishing signatures for accumulator" yellow
69 $BIN cmd 1 dhtput utente1 signature s '{"utente1":"HwctHmA3es
       kXqU8XZvZ5UYOuv6kLyPvfk440kpfXQXaXtjhb/CQuzg+bUscIQc9vNk7e
       XPH46Xkmq7ICQ8dnePw=","utente2":"IPyAistijtTqgXLTmC1Z1w4e
       r5EZAED1CRGDScfV+uQbImH4p3agJ8xHsD/OiyCdclevSo2kFCDBE0Haep
       IiSYM="}' utente1 0
70 $BIN cmd 1 dhtput utente2 signature s '{"utente1":"H8EmLQjjAw
       jtYzSldmKi/SqCx4wBvfxqZsCW9Td4+GFdCG1BQluI293q4WEPZMtZnAfi
       NBwbyfZx40t80bXTxpo=","utente2":"IPrI4rpbaoVuuDu4hMAJ8m9B8
       ec0pxm5XjCEkbiwErnBqSn1tPzBTS82o8i+qh2mNjEpwNjiQR9/g6cKdII
       rfhI="}' utente2 0
71 sleep 2
72
73 $BIN cmd 1 newpostmsg utente1 2 \"txt:Nice_Cross-Body_Bag\#im
       g:1.jpg\#title:The_Sak_Silverlake\"
74 $BIN cmd 1 newpostmsg utente1 1 \"txt:Nice_Cross-Body_Bag\#im
       g:2.jpg\#title:The_Rebecca_Minkoff\"
75 $BIN cmd 2 newpostmsg utente2 1 \"txt:Nice_Cross-Body_Bag\#im
       g:3.jpg\#title:The_Angelica_Klein\"
76 $BIN cmd 2 newpostmsg utente2 2 \"txt:Nice_Cross-Body_Bag\#im
       g:4.jpg\#title:The_Aldo_Anakardo\"
77 $BIN cmd 3 newpostmsg utente3 1 \"txt:Nice_Cross-Body_Bag\#im
       g:5.jpg\#title:The_Loeffler_Randall\"
78 $BIN cmd 1 follow utente1 [\"utente1\",\"utente2\",\"utente3\
       "]
79 $BIN cmd 2 follow utente2 [\"utente1\",\"utente2\"]
80 $BIN cmd 3 follow utente3 [\"utente1\",\"utente2\"]
81 $BIN cmd 1 dhtput utente1 home s [\"utente1\",\"utente2\"] ut
       ente1 0
82
83 coloredEcho "\nPress a button to publish the new accumulator
       with related signatures" blue
84 read -n1
85 $BIN cmd 1 dhtput utente1 order s [\"utente1\",\"utente2\",\"
       utente3\"] utente1 1
86 sleep 2
87 $BIN cmd 1 sendrawtransaction 0100000000009085f61646d696e5
       f3208077574656e746532212007eedc174ba9061088bc764189ecb85
       d351a615f0ca5781c77436735f23224c5afed0000
88 coloredEcho "\nWait for the transaction to be included in the
        blockchain" red
89 read -n1
90 $BIN cmd 3 newpostmsg utente3 1 \"txt:Nice_Cross-Body_Bag\#im
       g:5.jpg\#title:The_Loeffler_Randall\"
```

```
91 $BIN cmd 1 follow utente1 [\"utente1\",\"utente2\",\"utente3\
    "]
92
93 echo "done!"
```

Listing 5.1: Script per far partire la demo.

Una versione modificata di questo script da eseguire sulle macchine virtuali è presente in appendice. Qui verrà presentata la versione in locale, e verranno evidenziate le differenze tra i due quando necessario. Lo script aLaunch è preso direttamente dall'originale Accumunet e riportato in appendice, il suo scopo è quello di gestire le operazioni più comuni come la creazione e la cancellazione dei nodi.

Per prima cosa lo script avvia i 3 client e copia l'interfaccia web nella cartella data dei rispettivi client, in modo che il web server di ogni client possa presentare le pagine correttamente. Sulle macchine virtuali questo non è necessario, la cartella è già copiata ed è sufficiente avviare il client. Oltre all'interfaccia web viene fatta una copia del file twisterwallet.dat[1], che contiene le identità dei tre utenti coinvolti nella demo, e il file bootstrap.dat, che contiene i primi blocchi della block chain. Tra le transazioni già incluse in questo file di bootstrap c'è anche quella del primo accumulatore.

Alle righe 41-43 avviene il collegamento tra i nodi. Seguono poi una serie di comandi addwitnesstouser il cui compito è quello di memorizzare nel client i witness degli utenti. I witness sono memorizzati in memoria, ma in un client reale potrebbero essere recuperati dalla rete DHT. Viene quindi avviato il mining su un client che inizia a generare blocchi, e viene inserita nella rete la prima transazione che contiene la struttura. Bisogna a questo punto attendere che il client sblocchi la rete DHT, evento segnalato dalla comparsa nei log del messaggio B̈lockchain is now up-to-date, unpausing libtorrent session:.
In seguito a questo evento, nelle linee 69-70 sono pubblicate le liste di firme per l'accumulatore 5.5, sia per quello corrente sia per quello che verrà pubblicato in seguito. Dalla riga 73 alla riga 77 sono poi pubblicati i prodotti dei singoli utenti tramite il comando dhtput. La pubblicazione verrà rifiutata al terzo utente, come si può osservare nei log, in quanto non accumulato ancora. Nelle righe 78-80 ogni utente si iscrive ad almeno un torrent di un altro utente, per riceverne i prodotti pubblicati. Questo è equivalente a un follow

[1] In questo senso è stato impiegata una scorciatoia imppiegata anche dall'originale Accumunet, ovvero per semplificare le operazioni ogni client può loggarsi con tutti e tre gli utenti. In uno scenario reale ci sarebbero tre file twisterwallet.dat separati e differenti ognuno contenente una identità.

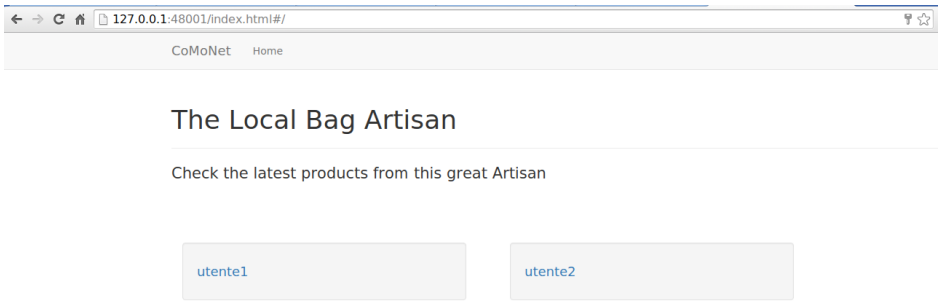

Figura 5.1: Schermata home.

sulla rete Twister, che a sua volta richiama il follow di Twitter. Arrivati a questo punto la situazione è quella osservabile negli screenshot 5.1 5.2.

Alla riga 83 viene pubblicata una nuova transazione contenente un accumulatore che autentica anche l'utente numero tre. Dopo la pubblicazione bisogna attendere qualche secondo affinché la transazione sia propagata a tutti e tre i client. Le firme erano invece state già pubblicate in precedenza. Dopo la propagazione viene pubblicato un prodotto per l'utente 3, e viene fatto in modo che l'utente 1 sia iscritto alla sua bacheca, ottenendo il risultato nelle figure 5.3 5.4.

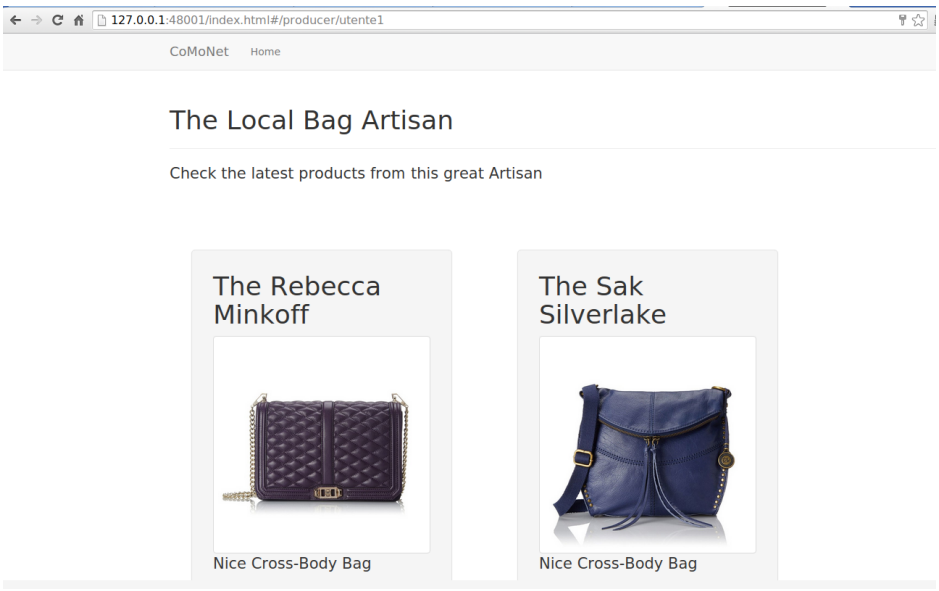

Figura 5.2: Pagina del primo utente con relativi prodotti.

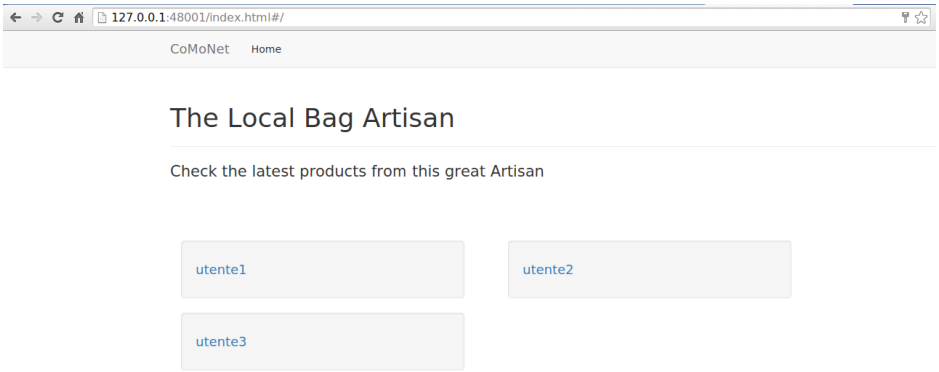

Figura 5.3: Home page dopo l'aggiornamento dell'accumulatore.

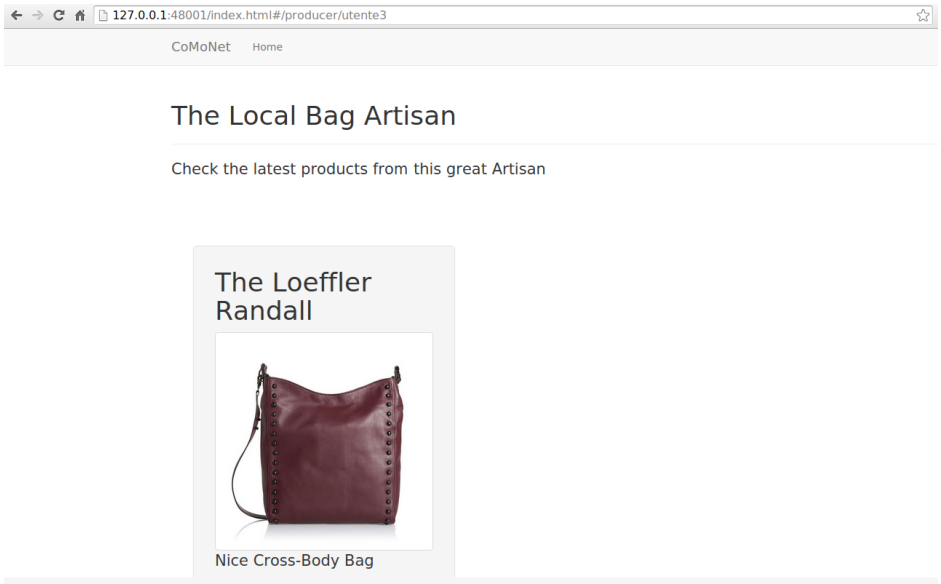

Figura 5.4: Prodotti dell'utente 3.

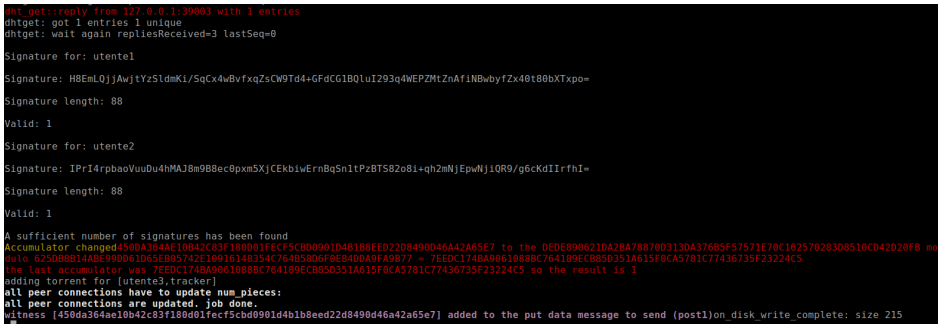

Figura 5.5: Recupero e controllo delle firme per validare l'accumulatore.

Capitolo 6

Conclusioni

Partendo dalla necessità di creare una rete peer-to-peer in cui fosse possibile applicare una moderazione, si sono esaminate strutture e soluzioni reputate adatte allo scopo, in particolare sistemi peer-to-peer come ZeroNet e OpenBazaar. Alla fine si è deciso di estendere il software Accumunet, un derivato di Twister, un social network che unisce le reti BitTorrent, DHT e Bitcoin per creare un clone di Twitter orientato all'anonimità degli utenti. Accumunet, creato come proof-of-concept all'Università di Genova nel 2014, è stato modificato per fare in modo che rispondesse in maniera più realistica a questo problema, creando un fork chiamato CoMoNet (Content Moderated Network). Per prima cosa è stato affrontato il problema della gestione dei permessi, in particolare la politica di aggiornamento dell'accumulatore crittografico. Si è deciso di adottare una soluzione ispirata dalle multisignatures di Bitcoin, in cui viene a crearsi una catena di accumulatori memorizzati nelle transazioni, e in cui ogni accumulatore è accompagnato da un indirizzo nella rete DHT che contiene una lista di firme che il client può usare per validarlo.
Si è poi trattato il problema della presentazione dei contenuti, in particolare creando una struttura memorizzata nella block chain che i client possono usare come riferimento per mostrare i contenuti proposti dai fornitori.
Si sono testate queste due soluzioni su una rete composta da 3 macchine virtuali, provando prima l'aggiornamento di un accumulatore e la verifica delle relative firme nella rete DHT, e poi la visualizzazione della struttura, in questo caso semplificata e implementata come un file json in cui è indicato l'ordine di visualizzazione dei fornitori.
Si sono poi discusse possibili estensioni a questo sistema. La più concreta è rappresentata dalle *threshold signatures*, che permetterebbero di memorizzare una firma a soglia unica nella block chain, a differenza della soluzione adottata in questo lavoro di memorizzare una lista di firme nella rete DHT.

Le altre estensioni esaminate sono invece di respiro più ampio, come la possibilità di realizzare un web distribuito con software simili a CoMoNet.
Si è evidenziato comunque che prima che software simili a CoMoNet possano raggiungere un mercato ampio è necessario un deciso sforzo nella direzione di migliorare l'usabilità.

Appendice A

Codice

Il codice completo di CoMoNet si trova all'indirizzo: `https://github.com/manuel-zulian/CoMoNet`.

```
1  int updateAccumulator() {
2      CTransaction previousAccumulator;
3    CTransaction txAccumulator;
4    uint256 acc_hashBlock;
5
6      // Recupera la prima transazione
7      string next_try;
8      next_try = admin_str + itostr(1);
9      printf( YELLOW "\n%s\n", next_try.c_str());
10     if (!GetTransaction(next_try, txAccumulator, acc_hashBloc
       k)) {
11         printf( RED "\nCan't retrieve the first accumulator, a
       borting\n" RESET "\n");
12         return ACC_ERROR;
13     }
14
15     for(int i = 2; i < 1024; i++) {
16         next_try = admin_str + itostr(i);
17
18         CTransaction temp;
19         uint256 temp2;
20
21         // Se non lo trova \'{e} arrivato all'ultimo, interro
       mpi.
22         if (!GetTransaction(next_try, temp, temp2)) {
23             printf( YELLOW "\nAdmin transaction %s not found\
       n", next_try.c_str());
24             break;
25         } else {
26             previousAccumulator = txAccumulator;
```

```
27              txAccumulator = temp;
28          }
29      }
30
31      string dht_address;
32      dht_address = boost::algorithm::unhex(txAccumulator.pubKe
        y.ToString());
33
34      string dht_address_previous_accumulator;
35      if(next_try == "_admin_2") {
36          dht_address_previous_accumulator = boost::algorithm::
        unhex(txAccumulator.pubKey.ToString());
37      } else {
38          dht_address_previous_accumulator = boost::algorithm::
        unhex(previousAccumulator.pubKey.ToString());
39      }
40
41      /**
42       * Recupera il numero di firme necessarie, cio\'{e} me
        t\'{a} degli utenti
43       * nell'accumulatore precedente + 1. Se \'{e} la prima tr
        ansazione, serve l'unanimit\'{a}.
44       * */
45      int needed_signatures;
46      printf( YELLOW "\nSearching signatures at address: %s\n",
        dht_address_previous_accumulator.c_str());
47      if(next_try == "_admin_2") {
48          printf( YELLOW "\nFirst transaction\n");
49          needed_signatures = computeNeededSignatures(dht_addre
        ss_previous_accumulator, true);
50      } else {
51          printf( YELLOW "\nNot first transaction\n");
52          needed_signatures = computeNeededSignatures(dht_addre
        ss_previous_accumulator, false);
53      }
54      printf( YELLOW "\nNeeds %i valid signatures to accept the
        current accumulator\n", needed_signatures);
55
56      /**
57       * Recupera le firme dalla rete dht.
58       */
59      Array p;
60      p.push_back(dht_address);
61      p.push_back("signature");
62      p.push_back("s");
63      Array result = dhtget(p, false).get_array();
64
65      if(!result.size()) {
66          printf( RED "\nNo signatures found\n");
```

```
67          return ACC_ERROR;
68      }
69
70      int validated_signatures = 0;
71
72      // Verifica le firme.
73      for( size_t i = 0; i < result.size(); i++ ) {
74          if( result.at(i).type() != obj_type )
75              continue;
76          Object resDict = result.at(i).get_obj();
77
78          BOOST_FOREACH(const Pair& item, resDict) {
79              if( item.name_ == "p" && item.value_.type() == ob
    j_type ) {
80                  Object pDict = item.value_.get_obj();
81                  BOOST_FOREACH(const Pair& pitem, pDict) {
82                      if( pitem.name_ == "v" && pitem.value_.ty
    pe() == obj_type ) {
83                          Object signatures = pitem.value_.get_
    obj();
84
85                          //printf( YELLOW "\narrivo qua");
86
87                          BOOST_FOREACH(const Pair& signature_p
    air, signatures) {
88                              string username = signature_pair.na
    me_;
89                              string signature = signature_pair.v
    alue_.get_str();
90
91                              printf( YELLOW "\nSignature for: %
    s\n", username.c_str());
92                              printf( YELLOW "\nSignature: %s\n",
     signature.c_str());
93                              printf( YELLOW "\nSignature length:
     %i\n", signature.size());
94
95                              Array temp;
96                              temp.push_back(username);
97                              temp.push_back(signature);
98                              string uppercase_acc = boost::to_up
    per_copy(txAccumulator.accumulator.ToString());
99                              temp.push_back(uppercase_acc);
100
101                             // Qui avviene l'effettiva validazi
    one
102                             Value validate = verifymessage(tem
    p, false);
103                             bool valid = validate.get_bool();
```

```
104                                 printf( YELLOW "\nValid: %d\n", val
       id);
105
106                                   if(valid) {
107                                       validated_signatures++;
108                                   }
109                               }
110                           }
111                       }
112                   }
113               }
114           }
115
116           if(validated_signatures < needed_signatures) {
117               return ACC_ERROR;
118           } else {
119               printf( YELLOW "\nA sufficient number of signatures h
       as been found\n");
120           }
121
122         mp_int new_acc;
123         mp_init(&new_acc);
124
125         std::vector< std::vector<unsigned char> > vAccData;
126         if( txAccumulator.accumulator.ExtractPushData(vAccData) ) {
127           std::vector<unsigned char> vch = vAccData[0];
128           if (mp_iszero(&last_accumulator_cache))
129             mp_init(&last_accumulator_cache);
130           mp_read_unsigned_bin(&new_acc, vch.data(), vch.end()-vch.
       begin());
131           if (mp_cmp(&last_accumulator_cache, &new_acc) == MP_EQ) {
132                   printf( YELLOW "Accumulator unchanged");
133             return ACC_UNCHANGED;
134           } else {
135                   printf( YELLOW "Accumulator changed");
136             mp_copy(&new_acc, &last_accumulator_cache);
137             return ACC_CHANGED;
138           }
139         } else {
140           printf( RED "Can't decode the accumulator, aborting" RESE
       T "\n");
141           return ACC_ERROR;
142         }
143       }
144
145       /**
146        * Calcola il numero di firme necessarie a considerare valido
              il prossimo accumulatore.
147        * @param dht_address L'indirizzo a cui recuperare le firme d
```

```
       ell'accumulatore precedente.
148    * @return Il numero di firme necessario a validare l'accumul
       atore.
149    */
150   int computeNeededSignatures(string dht_address, bool isFirst)
       {
151       Array p;
152       p.push_back(dht_address);
153       p.push_back("signature");
154       p.push_back("s");
155       Array result = dhtget(p, false).get_array();
156
157       if(!result.size()) {
158           printf( RED "\nNo signatures found\n");
159       }
160
161       int needed = 0;
162       int temp = 0;
163
164       for( size_t i = 0; i < result.size(); i++ ) {
165           if( result.at(i).type() != obj_type )
166               continue;
167           Object resDict = result.at(i).get_obj();
168
169           BOOST_FOREACH(const Pair& item, resDict) {
170               if( item.name_ == "p" && item.value_.type() == ob
       j_type ) {
171                   Object pDict = item.value_.get_obj();
172                   BOOST_FOREACH(const Pair& pitem, pDict) {
173                       if( pitem.name_ == "v" && pitem.value_.ty
       pe() == obj_type ) {
174                           Object signatures = pitem.value_.get_
       obj();
175
176                           BOOST_FOREACH(const Pair& signature_p
       air, signatures) {
177                               temp++;
178                               printf( YELLOW "\nIncrement numbe
       r of sig\n");
179                           }
180                       }
181                   }
182               }
183           }
184       }
185
186       /* Se \`{e} la prima transazione richiedi l'unanimit\`{a}
       */
187       if(!isFirst) {
```

```
188         needed = ceil(temp / 2) + 1;
189     } else {
190         needed = temp;
191     }
192
193     return needed;
194 }
```

Listing A.1: Funzione per il recupero dell'accumulatore e la verifica delle firme.

```
1  getStructure = function (cbFunc) {
2      var resultArg = null, errorArg = null;
3      twisterRpc("getstructure", [], function (resultArg, resul
       t) {
4              var structure;
5              console.log(result);
6              // Si suppone che la struttura sia una json benfo
       rmata.
7              structure = JSON.parse(result);
8              console.log(structure);
9
10             // Per non intasare la blockchain, l'ordine in cu
       i mostrare gli utenti va a prenderlo in dht.
11             twisterRpc("dhtget", [structure.order, "order", "
       s"], function (resultArgs, result) {
12                     console.log(result);
13                     structure.order = result[0].p.v;
14                     console.log(structure);
15                     cbFunc(structure);
16                 }, resultArg,
17                 function (errorArg, error) {
18
19                 }, errorArg);
20         }, resultArg,
21         function (errorArg, error) {
22             console.log(error);
23         }, errorArg);
24 };
```

Listing A.2: Funzione javascript per il recupero dei dati dal client CoMoNet.

```
1          var fillPosts = function (posts) {
2                  $scope.posts = [];
3                  posts.forEach(function(item){
4                      var post = {};
5                      var temp = item.userpost.msg.split("#");
```

```
6                       var temp2 = temp[0].split(":");
7                       post.text = temp2[1].replace(/_/g, " ");
8                       var temp3 = temp[1].split(":");
9                       post.img = temp3[1].replace(/^"|"$/, "");
10                      var temp4 = temp[2].split(":");
11                      post.title = temp4[1].replace(/_/g, " ").
    replace(/^"|"$/, "");
12                      $scope.posts.push(post);
13                  });
14                  //$scope.posts = posts;
15              }
```

Listing A.3: Funzione javascript per il popolamento della pagina dei prodotti di un utente.

```
1   main.controller('DashboardController', function ($scope,
    main_state, $routeParams, rpcQuery, $interval) {
2       var fillProducers = function (bulk) {
3           var producersArray = bulk.value;
4           $scope.producers = producersArray;
5       };
6
7       rpcQuery.getstructure().then(function (structure) {
8           $scope.rows = structure.rows;
9           $scope.columnswidth = 12 / structure.columns;
10          $scope.producers = structure.order;
11      });
12  });
```

Listing A.4: Controller della home page.

Appendice B

Configurazione di test

Per testare il funzionamento è stato usato il software di virtualizzazione VirtualBox. É stata configurata una macchina con due processori logici, 2 GB di memoria, due interfacce di rete e controller USB 1.1 OHCI[1]. Sono inoltre state abilitate le estensioni di virtualizzazione della CPU dell'host. Alla macchina è stato assegnato un disco virtuale da 8 GB, allocato dinamicamente in formato VMDK. Sia il quantitativo di memoria primaria sia quello di memoria secondaria sono sovradimensionati rispetto alle necessità di esecuzione di CoMoNet, ma sono corretti per le necessità di compilazione. Le impostazioni non dichiarate sono quelle generate di default dalla versione 5.0.2 di VirtualBox.
Le interfacce di rete sono state configurate nel seguente modo:

- **Scheda 1** Network di tipo bridged a un'interfaccia dell'host dotata di connessione a Internet.

- **Scheda 2** Network solo host collegato a un'interfaccia virtuale creata da VirtualBox (vboxnet0), con indirizzi IP assegnati manualmente (senza DHCP) nella sotto-rete 192.168.56.0/24.

Sulla macchina virtuale è stata installata un'immagine base di Debian versione 8.1 per architettura a 64 bit, in particolare in formato net-installer. Sono state seguite tutte le scelte di default durante l'installazione, se non per quelle relative a lingua e layout della tastiera. Si è creato un utente root con password *accumunet* e un utente accumunet con identica password.

[1]Questa impostazione è stata cambiata per prevenire problemi che si sono presentati al momento di clonare la macchina virtuale.

Sulla macchina virtuale sono stati installati per prima cosa i seguenti pacchetti:

- zsh
- git
- autoconf
- libtool
- build-essential
- libboost-all-dev
- libssl-dev
- libdb++-dev
- libminiupnpc-dev
- automake

Con l'utente standard è stato quindi clonato il repository git all'indirizzo `https://github.com/manuel-zulian/accumunet` nella cartella accumunet all'interno della home. Per la compilazione si sono quindi dati i comandi ./bootstrap.sh e make[2] all'interno della cartella scaricata. L'eseguibile in output conserva il nome twisterd, dal software originale di cui questo è il fork.
A questo punto si è creata una cartella data nella home in cui conservare i dati di esecuzione di CoMoNet[3]. Si è proceduto quindi a clonare la macchina virtuale due volte, avendo cura di reinizializzare l'indirizzo MAC delle interfacce di rete e l'IP statico assegnato, assegnando gli indirizzi 192.168.56.2, 192.168.56.3, 192.168.56.4 alle tre macchine come in configurazione B.1.

[2]La compilazione di questo software può essere velocizzata, soprattutto la prima volta, usando il flag -j di make, avviando di fatto j compilatori in parallelo. Questo flag necessità però non solo di più processori logici, ma anche di più memoria primaria.

[3]Passaggio non strettamente necessario, in quanto di default i dati sarebbero salvati nella cartella .twister all'interno della home

```
1  source /etc/network/interfaces.d/*
2
3  # The loopback network interface
4  auto lo
5  iface lo inet loopback
6
7  # The primary network interface
8  allow-hotplug eth0
9  iface eth0 inet dhcp
10
11 auto eth1
12 iface eth1 inet static
13 address 192.168.56.2
14 netmask 255.255.255.0
15
```

Listing B.1: Esempio di configurazione del file /etc/network/interfaces per una delle macchine.

Per avviare CoMoNet su ognuna delle tre macchine sono chiamati due script. Il primo B.2 avvia CoMoNet, il secondo B.3 collega due macchine tra loro e deve essere eseguito dopo che tutti e tre i client CoMoNet sono stati avviati.

```
1  #!/bin/bash
2
3  BASEDIR=/home/accumunet
4
5  $BASEDIR/accumunet/twisterd -rpcuser=user -rpcpassword=pwd -r
       pcallowip=192.168.56.1 -htmldir=$BASEDIR/accumunet/html -d
       atadir=$BASEDIR/data -daemon
6
```

Listing B.2: Script per l'avvio di CoMoNet.

```
1  #!/bin/bash
2
3  ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
       rd=pwd addnode 192.168.56.3:28333 add
4
```

Listing B.3: Esempio di script per il collegamento di due macchine.

Dopo il collegamento dei tre client, si può utilizzare lo script incluso nella macchina principale B.4 per effettuare una prova di funzionamento della rete.

Listing B.4: Script per demo.

```
#!/bin/bash

function coloredEcho(){
  local exp=$1;
  local color=$2;
  if ! [[ $color =~ '^[0-9]$' ]] ; then
  case $(echo $color | tr '[:upper:]' '[:lower:]') in
  black) color=0 ;;
  red) color=1 ;;
  green) color=2 ;;
  yellow) color=3 ;;
  blue) color=4 ;;
  magenta) color=5 ;;
  cyan) color=6 ;;
  white|*) color=7 ;; # white or invalid color
  esac
  fi
  tput setaf $color;
  echo -e $exp;
  tput sgr0;
}

sleep 5

~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
    rd=pwd addnode 192.168.56.3:28333 add
~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
    rd=pwd addnode 192.168.56.2:28333 add
~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
    rd=pwd addnode 192.168.56.3:28333 add

~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
    rd=pwd addwitnesstouser utente1 357d626250e41eb0390d60e4
    c33859369681fec371ab532e428b63059e97c6a7
~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
    rd=pwd addwitnesstouser utente1 357d626250e41eb0390d60e4
    c33859369681fec371ab532e428b63059e97c6a7
~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
    rd=pwd addwitnesstouser utente1 357d626250e41eb0390d60e4
    c33859369681fec371ab532e428b63059e97c6a7
sleep 0.5
~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
    rd=pwd addwitnesstouser utente2 26ff76d0f66b8403bda6534fc
    e3dbcaf47ffd5eb7e06720dd3d58aee587a39a5
~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
    rd=pwd addwitnesstouser utente2 26ff76d0f66b8403bda6534fc
    e3dbcaf47ffd5eb7e06720dd3d58aee587a39a5
```

```
35 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd addwitnesstouser utente2 26ff76d0f66b8403bda6534fc
      e3dbcaf47ffd5eb7e06720dd3d58aee587a39a5
36 sleep 0.5
37 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd addwitnesstouser utente3 450da364ae10b42c83f180d01f
      ecf5cbd0901d4b1b8eed22d8490d46a42a65e7
38 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd addwitnesstouser utente3 450da364ae10b42c83f180d01f
      ecf5cbd0901d4b1b8eed22d8490d46a42a65e7
39 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd addwitnesstouser utente3 450da364ae10b42c83f180d01f
      ecf5cbd0901d4b1b8eed22d8490d46a42a65e7
40 sleep 0.5
41 coloredEcho "starting miner..." yellow
42 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd setgenerate true 1
43 sleep 0.5
44 coloredEcho "Creating structure" yellow
45 # Transazione di struttura
46 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd sendrawtransaction 01000000000d0c5
      f737472756374757265 5f3108077574656e74653129287b22726
      f7773223a332c22636f6c756d6e73223a332c226f72646572223
      a227574656e746531227dc59b0000
47 coloredEcho "\nwait a moment for the dht to be ready" red
48 sleep 5
49 coloredEcho "\nPress any button when the dht is loaded" blue
50 read -n1
51 coloredEcho "Publishing order for structure" yellow
52 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd dhtput utente1 order s '["utente1","utente2"]' uten
      te1 0
53 coloredEcho "Publishing signatures for accumulator" yellow
54 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd dhtput utente1 signature s '{"utente1":"HwctHmA3esk
      XqU8XZvZ5UYOuv6kLyPvfk440kpfXQXaXtjhb/CQuzg+bUscIQc9vNk7eX
      PH46Xkmq7ICQ8dnePw=","utente2":"IPyAistijtTqgXLTmC1Z1w4er5
      EZAEDlCRGDScfV+uQbImH4p3agJ8xHsD/OiyCdclevSo2kFCDBE0HaepIi
      SYM="}' utente1 0
55 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd dhtput utente2 signature s '{"utente1":"H8EmLQjjAwj
      tYzSldmKi/SqCx4wBvfxqZsCW9Td4+GFdCG1BQluI293q4WEPZMtZnAfiN
      BwbyfZx40t80bXTxpo=","utente2":"IPrI4rpbaoVuuDu4hMAJ8m9B8e
      c0pxm5XjCEkbiwErnBqSn1tPzBTS82o8i+qh2mNjEpwNjiQR9/g6cKdIIr
      fhI="}' utente2 0
56 sleep 2
57 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd newpostmsg utente1 1 \"Primo_post_utente1\"
```

```
58 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd newpostmsg utente1 2 \"Vendo_vino_buono\"
59 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd newpostmsg utente2 1 \"Ciao_sono_utente2\"
60 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd newpostmsg utente2 2 \"Vendo_vino_buonissimo\"
61 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd newpostmsg utente3 1 \"test_di_messaggio\"
62 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd follow utente1 [\"utente1\",\"utente2\",\"utente3\"
      ]
63 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd follow utente2 [\"utente1\",\"utente2\"]
64 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd follow utente3 [\"utente1\",\"utente2\"]
65 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd dhtput utente1 home s [\"utente1\",\"utente2\"] ute
      nte1 0
66
67 coloredEcho "\nPress a button to publish the new accumulator
      with related signatures" blue
68 read -n1
69 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd dhtput utente1 order s [\"utente1\",\"utente2\",\"u
      tente3\"] utente1 1
70 sleep 2
71 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd sendrawtransaction 0100000000009085f61646d696e5
      f3208077574656e746532212007eedc174ba9061088bc764189ecb85
      d351a615f0ca5781c77436735f23224c5afed0000
72 coloredEcho "\nWait for the transaction to be included in the
       blockchain" red
73 read -n1
74 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd newpostmsg utente3 1 \"test_di_messaggio\"
75 ~/accumunet/twisterd -genproclimit=1 -rpcuser=user -rpcpasswo
      rd=pwd follow utente1 [\"utente1\",\"utente2\",\"utente3\"
      ]
76
77 echo "done!"
```

Bibliografia

[1] M. Nottingham, "Will there be a distributed http?." `https://www.mnot.net/blog/2015/08/18/distributed_http`. Accesso: 2015-09-05.

[2] C. Zhang, J. Sun, X. Zhu, and Y. Fang, "Privacy and security for online social networks: challenges and opportunities," *Network, IEEE*, vol. 24, no. 4, pp. 13–18, 2010.

[3] C. H. Ding, S. Nutanong, and R. Buyya, "P2p networks for content sharing," *arXiv preprint cs/0402018*, 2004.

[4] C. Zhu, *Streaming Media Architectures, Techniques, and Applications: Recent Advances: Recent Advances, pp. 341*. IGI Global, 2010.

[5] D. Yin, Z. Xue, L. Hong, B. D. Davison, A. Kontostathis, and L. Edwards, "Detection of harassment on web 2.0," *Proceedings of the Content Analysis in the WEB*, vol. 2, pp. 1–7, 2009.

[6] Z. Lu, Y. Wang, and Y. R. Yang, "An analysis and comparison of cdn-p2p-hybrid content delivery system and model," *Journal of Communications*, vol. 7, no. 3, pp. 232–245, 2012.

[7] "Openbazaar." `https://github.com/OpenBazaar/OpenBazaar`. Accesso: 2015-09-05.

[8] M. Freitas, "twister-a p2p microblogging platform," *arXiv preprint arXiv:1312.7152*, 2013.

[9] A. Passaglia, "Un social network moderato e distribuito," Master's thesis, Università di Genova, 7 2014.

[10] M. El Dick and E. Pacitti, "Content distribution in p2p systems," 2009.

[11] "Openbazaar - building a decentralized marketplace network." `http://www.slideshare.net/openbazaar/open-bazaar`. Accesso: 2015-09-03.

[12] "Ricardian contracts in openbazaar." https://gist.github.com/drwasho/a5380544c170bdbbbad8. Accesso: 2015-09-03.

[13] "Bitcoin transactions." https://en.bitcoin.it/wiki/Transaction. Accesso: 2015-09-05.

[14] "A pseudonymous trust system for a decentralized anonymous marketplace." https://gist.github.com/dionyziz/e3b296861175e0ebea4b. Accesso: 2015-09-03.

[15] L. Wang and J. Kangasharju, "Real-world sybil attacks in bittorrent mainline dht," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, pp. 826–832, IEEE, 2012.

[16] J. R. Douceur, "The sybil attack," in *Peer-to-peer Systems*, pp. 251–260, Springer, 2002.

[17] "Zeronet - decentralized web platform using bitcoin cryptography and bittorrent network." https://docs.google.com/presentation/d/1_2qK1IuOKJ51pgBvllZ9Yu7Au2l551t3XBgyTSvilew/pub?start=false&loop=false&delayms=3000&slide=id.g9a1cce9ee_8_21. Accesso: 2015-09-02.

[18] "Does zeronet uses bitcoin's blockchain?." http://zeronet.readthedocs.org/en/latest/faq/#does-zeronet-uses-bitcoins-blockchain. Accesso: 2015-09-02.

[19] "Twister." http://twister.net.co/?page_id=16. Accesso: 2015-09-05.

[20] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Consulted*, vol. 1, no. 2012, p. 28, 2008.

[21] J. Benaloh and M. De Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *Advances in Cryptology—EUROCRYPT'93*, pp. 274–285, Springer, 1994.

[22] N. Fazio and A. Nicolosi, "Cryptographic accumulators: Definitions, constructions and applications," *Paper written for course at New York University: www. cs. nyu. edu/nicolosi/papers/accumulators. pdf*, 2002.

[23] K. Nyberg, "Fast accumulated hashing," in *Fast Software Encryption*, pp. 83–87, Springer, 1996.

[24] H. de Meer, M. Liedel, H. C. Pöhls, J. Posegga, and K. Samelin, "Indistinguishability of one-way accumulators," tech. rep., Technical Report MIP-1210, Faculty of Computer Science and Mathematics (FIM), University of Passau, 2012.

[25] J. Li, N. Li, and R. Xue, "Universal accumulators with efficient non-membership proofs," in *Applied Cryptography and Network Security*, pp. 253–269, Springer, 2007.

[26] C. Fromknecht, D. Velicanu, and S. Yakoubov, "Certcoin: A namecoin based decentralized authentication system," tech. rep., Technical Report. Massachusetts Institute of Technology, MA, USA. 6.857 Class Project, 2014.

[27] C. Fromknecht, D. Velicanu, and S. Yakoubov, "A decentralized public key infrastructure with identity retention," tech. rep., Cryptology ePrint Archive, Report 2014/803, 2014. http://eprint. iacr. org, 2014.

[28] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Advances in Cryptology—CRYPTO 2002*, pp. 61–76, Springer, 2002.

[29] T. Locher, D. Mysicka, S. Schmid, and R. Wattenhofer, "Poisoning the kad network," in *Distributed Computing and Networking*, pp. 195–206, Springer, 2010.

[30] R. Fantacci, L. Maccari, M. Rosi, L. Chisci, L. M. Aiello, and M. Milanesio, "Avoiding eclipse attacks on kad/kademlia: an identity based approach," in *Proceedings of the 2009 IEEE international conference on Communications*, pp. 983–987, IEEE Press, 2009.

[31] "Twoofthree.sh." `https://gist.github.com/gavinandresen/3966071`. Accesso: 2015-09-01.

[32] "What are the limits of m and n in m-of-n multisig addresses?." `http://bitcoin.stackexchange.com/questions/23893/what-are-the-limits-of-m-and-n-in-m-of-n-multisig-addresses`. Accesso: 2015-08-30.

[33] S. Goldfeder, J. Bonneau, E. W. Felten, J. A. Kroll, and A. Narayanan, "Securing bitcoin wallets via threshold signatures." `http://www.cs.princeton.edu/~stevenag/bitcoin_threshold_signatures.pdf`, 2014.

[34] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[35] D. W. Kravitz, "Digital signature algorithm," July 27 1993. US Patent 5,231,668.

[36] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[37] P. MacKenzie and M. K. Reiter, "Two-party generation of dsa signatures," in *Advances in Cryptology—CRYPTO 2001*, pp. 137–154, Springer, 2001.

[38] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in cryptology—EUROCRYPT'99*, pp. 223–238, Springer, 1999.

[39] J. Benet, "Ipfs - content addressed, versioned, p2p file system (draft 3)." `https://github.com/ipfs/ipfs/blob/master/papers/ipfs-cap2pfs/ipfs-p2p-file-system.pdf?raw=true`. Accesso: 2015-09-05.

Ringraziamenti

Ringrazio prima di tutto i miei genitori Michela e Leonildo, che mi hanno supportato in tutti i modi possibili.
Poi Laura, una ragazza per me unica a cui voglio molto bene.
Un grazie a Mirka che mi sopporta da tanto tempo quanto i miei genitori e a Davide, nella speranza che un giorno possa apprezzare.
Un grazie ad Alberto per avermi insegnato molto, non solo professionalmente.
Un ringraziamento al resto dei parenti, che sono pochi e posso citare tutti: la nonna Anna, il nonno Pietro, la zia Antonella e lo zio Flavio, e le mie cugine Jessica e Alessandra.
Grazie a Sebastiano, perché insieme a lui ho imparato quello che sui banchi di scuola non si poteva imparare.
Un grazie al mitico generale Badoglio, ovvero Bedeschi, a Niki, al nerd Aduso, alla Addi, Giulia, Martina E, Martina F, Clara, Federica e a tutti gli altri che ho conosciuto nel corso degli anni.
Un grazie ai colleghi con cui ho studiato per la laurea magistrale e con cui mi sono divertito: Mattia, Enrico, Giacomo, Elena, Pino (Valentino), Andrea e tutti gli altri che non elenco solo per non dilungarmi.
Un grazie a Enrico che ha portato pazienza mentre svolgevo la tesi e agli altri colleghi di lavoro.
Per finire, ringrazio i vari professori che si sono avvicendati nel corso degli anni, davvero troppi per citarli tutti, se non il mitico professor Brunelli e il mio relatore.