

UNIVERSITÀ DEGLI STUDI DI UDINE

---

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Triennale in Informatica

Tesi di Laurea

BITCOIN  
MONETA ELETTRONICA  
PEER-TO-PEER

./img/uniud\_lite-eps converted to .pdf

Relatore:  
Prof. IVAN SCAGNETTO

Laureando:  
MATTEO PAOLUZZI

---

ANNO ACCADEMICO 2012-2013



Ai miei genitori  
per non avermi tagliato i viveri,  
a Serena  
per il continuo e incessante supporto



---

## **Sommario**

Verrà descritta la struttura e il funzionamento della rete Bitcoin, un sistema monetario decentralizzato virtuale. Per prima cosa si procederà ad un raffronto tra le altre tipologie di reti P2P e la rete Bitcoin, evidenziandone le differenze e il perché tale rete sfugga ai normali criteri di catalogazione, pur rientrandone sotto alcuni punti di vista ben specifici. Verrà poi analizzata la rete nello specifico, illustrandone scopi, funzionamento, utilizzi e criticità, queste ultime soprattutto a confronto con le altre tipologie di rete nei casi attinenti. Infine verranno trattati in modo informale alcuni temi di carattere socio-economico collegati all'utilizzo di Bitcoin, analizzando brevemente alcune vicende di cronaca che negli ultimi anni hanno avuto tra i protagonisti tale rete.



---

## **Abstract**

English abstract





# Indice

<b>1</b>	<b>Panoramica generale sulle reti P2P</b>	<b>1</b>
1.1	Distribuzione dei file in una rete P2P . . . . .	2
1.1.1	Scalabilità ([?]) . . . . .	3
1.1.2	Ricerca di informazioni . . . . .	5
<b>2</b>	<b>Bitcoin: moneta elettronica decentralizzata</b>	<b>9</b>
2.1	Moneta Elettronica . . . . .	9
2.2	Portafogli e indirizzi . . . . .	10
2.3	Transazioni . . . . .	11
2.4	Timestamp e Proof-of-Work . . . . .	13
2.5	Network . . . . .	15
2.6	Risorse necessarie . . . . .	16
2.7	Gestione dei valori . . . . .	19
2.8	Analisi della rete . . . . .	21
2.8.1	Topologia . . . . .	21
2.8.2	Propagazione delle informazioni . . . . .	22
2.8.3	Informazioni scomparse . . . . .	24
2.9	Fork della blockchain . . . . .	26
2.9.1	Creazione del modello . . . . .	26
2.9.2	Misurazioni . . . . .	28
<b>3</b>	<b>Privacy</b>	<b>33</b>
3.1	Analisi quantitativa della privacy . . . . .	33
3.1.1	Il modello antagonistico . . . . .	34
3.1.2	Quantificazione della Privacy . . . . .	34
3.2	Applicare il modello . . . . .	39
3.2.1	Euristiche per sfruttare il client . . . . .	39
3.2.2	Analisi basata sul comportamento . . . . .	39
3.2.3	Simulazione: utilizzo di Bitcoin in ambiente universitario . . . . .	40
3.2.4	Risultati della simulazione . . . . .	42
3.3	Caso reale: analisi di un furto . . . . .	44
3.3.1	Mappare la rete Bitcoin . . . . .	44
3.3.2	Analisi di un furto . . . . .	48

---

3.4	Contromisure e consigli per aumentare la privacy . . . . .	50
<b>4</b>	<b>Sicurezza</b>	<b>53</b>
4.1	Reti P2P in genere . . . . .	53
4.1.1	Attacchi di Basso Livello . . . . .	53
4.1.2	Attacchi di Medio Livello . . . . .	55
4.1.3	Attacchi al livello P2P . . . . .	57
4.1.4	Contromisure generali . . . . .	59
4.2	La rete Bitcoin . . . . .	59
4.2.1	Fork della blockchain arbitrario . . . . .	59
4.2.2	Ridurre il forking: comunicazione tra i nodi . . . . .	62
4.2.3	Impedire riscritture della blockchain con lo scetticismo . . . . .	64
4.3	I Portafogli . . . . .	65
4.4	Bitcoin-Exchange . . . . .	67
<b>5</b>	<b>Mining Pools</b>	<b>69</b>
5.1	Funzionamento generico . . . . .	69
5.2	Sistemi di retribuzione semplici . . . . .	71
5.2.1	Proporzionale . . . . .	71
5.2.2	Pay-Per-Share (PPS) . . . . .	73
5.3	Sistemi di retribuzione a punteggio . . . . .	74
5.3.1	Metodo di Slush . . . . .	74
5.3.2	Metodo Geometrico . . . . .	75
5.3.3	Pay-per-lst-N-shares (PPLNS) . . . . .	78
5.4	Pay-per-share privo di rischi . . . . .	79
5.4.1	Maximum Pay-Per-Share (MPPS) . . . . .	79
5.4.2	Shared maximum Pay-Per-Share (SMPPS) . . . . .	80
5.4.3	Equalized shared maximum Pay-Per-Share (ESMPPS) . . . . .	81
5.5	Attacchi . . . . .	82
5.5.1	Pool-Hopping . . . . .	82
5.5.2	Sottrazione di blocchi . . . . .	83
5.5.3	Sabotaggi . . . . .	83
5.5.4	Imboscata . . . . .	83
5.5.5	Contromisura: Share Offuscate . . . . .	84
<b>6</b>	<b>Scripting</b>	<b>85</b>
6.1	Contratti . . . . .	90
6.1.1	Deposito temporaneo . . . . .	91
6.1.2	Acquisto di beni con mediatore . . . . .	92
6.1.3	Raccolta fondi assicurata . . . . .	93

---

<b>7</b>	<b>Argomenti correlati</b>	<b>95</b>
7.1	Statistiche . . . . .	95
7.2	Deflazione . . . . .	97
7.3	Esempi reali ???titolo da cambiare . . . . .	98
7.3.1	Distributore automatico . . . . .	98
7.3.2	Bancomat . . . . .	98
7.3.3	Silkroad ??FORSE?? . . . . .	98
<b>A</b>	<b>Simulatore Bitcoin per la valutazione della privacy dell'utente</b>	<b>99</b>
<b>B</b>	<b>Probabilità di successo per l'attaccante</b>	<b>101</b>



# Capitolo 1

## Panoramica generale sulle reti P2P

Negli ultimi anni si è assistito ad una progressiva alterazione delle leggi che regolano il mondo dell'hardware informatico in campo consumer: l'aumento della potenza di calcolo del singolo elaboratore non è più economicamente conveniente. Per fortuna la soluzione è stata immediata e consiste nello sfruttamento di più elaboratori collegati in rete che condividono (genericamente parlando) risorse. Sebbene per compiti specifici risulta spesso conveniente creare una rete ad-hoc, per l'utilizzo di tutti i giorni da parte dell'utente comune, la rete per eccellenza risulta essere senza ombra di dubbio la rete Internet. A causa di questa sua centralità, è stata scelta come base per lo sviluppo di applicazioni dedicate alla condivisione di risorse di vario tipo da parte di utenti con interessi in comune, creando di fatto una sottorete virtuale all'interno della già virtuale rete internet.

Questa condivisione di risorse da parte di utenti per un interesse comune definisce il nucleo di quelle che vengono chiamate **reti Peer-to-Peer**, da qui in avanti abbreviate come *reti P2P*. Data la grande diffusione di queste reti e i loro svariati obiettivi, è comprensibile che ci siano molti disaccordi sulla definizione esatta di *rete P2P*.

Una classificazione molto adatta agli scopi di questo documento è quella presente in [?] che distingue tre diversi livelli di rete:

1. **Infrastrutture P2P**, il cui scopo è porre le basi per i livelli successivi fornendo funzioni di comunicazione, integrazione e “traduzione” tra le varie componenti della rete. In particolare forniscono servizi che permettono la localizzazione e la comunicazione tra gli utenti (da ora in avanti **peers**) e l'identificazione, l'utilizzo e lo scambio delle risorse, oltre che l'implementazione delle politiche di sicurezza quali autenticazione e autorizzazione.
2. **Applicazioni P2P**, che utilizzano i servizi offerti dal livello di infrastruttura per offrire all'utente (qui inteso come essere umano interagente con la macchina, la quale è il peer vero e proprio) le funzionalità della rete. Sono in pratica le interfacce tra l'infrastruttura e la persona.

- 
3. **Fenomeni sociali** derivanti dall'utilizzo delle applicazioni P2P. Spesso infatti la forte coesione di intenti tra gli utenti di una rete P2P porta alla nascita di comunità virtuali, centri di aggregazione e correnti di pensiero che esulano dalla sfera prettamente informatica.

Come si vedrà più avanti nel corso della trattazione, la rete Bitcoin è caratterizzata da tutti e tre i livelli in modo molto più peculiare di altre reti P2P maggiormente diffuse e famose.

## 1.1 Distribuzione dei file in una rete P2P

La risorsa indubbiamente più abbondante e facilmente condivisibile è lo spazio di archiviazione, per questo le reti P2P in cui vengono condivisi file sono tra le più diffuse e variegate. La loro diffusione è talmente ampia che spesso l'intero concetto di rete P2P viene ridotto a quello di rete P2P per file-sharing, motivo per cui la classificazione delle reti P2P intese in senso generico spesso si fonda sul metodo di condivisione dei file, anche quando questo non è lo scopo della rete.

Indugiando in questa generalizzazione, studieremo un caso tipico: lo scaricamento di un file attraverso il modello classico client-server e alcune reti P2P ad ampia diffusione.

Prima di cominciare è necessario chiarire un concetto che spesso è causa di confusione: il modello P2P non è una alternativa al modello client-server, bensì una sua reimplementazione meno gravosa per il server.

Infatti, se nel modello client-server “puro” i ruoli sono definiti ed immutabili dall'inizio della comunicazione fino al suo completamento, quando si comunica in P2P i ruoli sono relativi al collegamento esistente tra i peer.

Prendiamo il caso in cui il peer **A** voglia ottenere dal peer **B** il file *file.txt*. All'inizio della comunicazione il peer **A** sarà il client e il peer **B** sarà invece il server. Immaginiamo che, mentre questo trasferimento è in atto, un terzo peer **C** voglia avere il file *file.txt*.

Se ci troviamo al di fuori di una rete P2P (ad esempio nel normale download di un file da internet tramite browser), i ruoli di **A** e **B** non subiscono variazioni e il peer **C** assume il ruolo di client del server **B**, il quale dovrà ottimizzare le sue risorse di banda e cpu per servire sia **A** che **C** contemporaneamente. NOTA:(da questo discorso esulano tematiche quali il multitasking della cpu: il punto di vista è quello di un ipotetico utente che osserva la macchina in tempo reale).

All'interno di una rete P2P invece, nel momento in cui comincia a scaricare *file.txt*, **C** è consapevole che **B** ne possiede una copia completa e **A** una parziale e, contemporaneamente, **A** verrà informato che **C** sta scaricando lo stesso file. Quello che succede è che **B** farà da server per **A** e **C** (i quali saranno i suoi client), mentre **A** e **C** si scambieranno le parti di *file.txt* che mancano l'uno all'altro (ovvero saranno sia client che server contemporaneamente). Il risultato è una sostanziale ottimizzazione delle risorse a disposizione nella rete e una maggiore velocità di “diffusione” del file.

---

Il linguaggio sopra adottato è forzatamente generico: ciò deriva dall'ampio numero di reti P2P per il filesharing esistenti, ognuna delle quali implementa a modo suo le casistiche di aggiunta/rimozione (*churn*) di un peer (chiamato solitamente **nodo** nell'ambito del file sharing) dalla rete, la ricerca dei file e il trasferimento dei contenuti, tutti comunque rispettando il procedimento generico sopra descritto.

### 1.1.1 Scalabilità ( [?])

Analizziamo la questione da un punto di vista più formale. Avremo bisogno dei seguenti dati.

$u_s$  frequenza di upload verso il server

$u_i$  frequenza di upload dell' $i$ -esimo peer

$d_i$  frequenza di download dell' $i$ -esimo peer

$F$  dimensione in bit del file da distribuire

$N$  numero di peer che vuole una copia del file

$D_{cs}$  tempo di distribuzione del file per l'architettura client-server

Per semplificare i conti senza invalidarne l'efficacia, assumiamo che la rete in esame sia priva di "disturbi" e sia dedicata esclusivamente allo scambio di file in esame senza altre comunicazioni passanti su di essa.

#### Caso Client-Server

Osservazioni:

- Il server deve trasmettere il file a  $N$  peer, quindi  $NF$  bit. Data la frequenza di upload  $u_s$ , il tempo per distribuire il file deve essere almeno  $NF/u_s$ .
- sia  $d_{min} = \min\{d_1, d_2, \dots, d_N\}$  la frequenza di download del peer con il valore più basso. Tale peer riceverà il file in almeno  $F/d_{min}$  secondi, che è quindi il tempo minimo di distribuzione.

Da cui

$$D_{cs} \geq \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{min}} \right\}$$

Questo è il limite inferiore al tempo di distribuzione minimo per l'architettura client-server. Trattiamo il caso ottimo e consideriamolo come il tempo di distribuzione effettivo, ovvero:

$$D_{cs} = \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{min}} \right\}$$

---

Da questa ultima espressione si vede come per  $N$  sufficientemente grande, il tempo di distribuzione è dato da  $NF/u_s$ , stabilendo quindi che esso aumenta linearmente all'aumentare del numero  $N$  dei peer.

### Caso P2P

La situazione cambia nel caso di architetture P2P, in cui ciascun peer assiste il server nella distribuzione del file. Dato che bisogna tenere conto di come ogni singolo peer distribuisce le sue porzioni di file, il calcolo risulta molto complesso. Possiamo però ottenere una semplice espressione del tempo minimo di distribuzione. A tale scopo bisogna fare alcune osservazioni:

- All'inizio dell'analisi solo il server possiede il file completo. È quindi necessario inviare ogni singolo bit del file all'interno della rete almeno una volta perché sia trasmesso alla comunità, quindi il minimo tempo di distribuzione è almeno  $F/u_s$ . La prima differenza è già visibile: dato che i peer ridistribuiscono i file, non è necessario che il server invii due volte lo stesso bit.
- Vediamo anche che il peer con la frequenza di download più bassa non può ottenere il file in meno di  $F/d_{min}$  secondi.
- Infine osserviamo come la capacità di upload della rete è quella del server più quella di ciascun peer. Denotando quest'ultimo con  $u_{tot}$  e sapendo che il numero totale di bit da trasferire è  $FN$ , possiamo stabilire che il tempo di distribuzione minimo è almeno  $FN/u_{tot}$ .

Unendo le affermazioni precedenti in una unica formulazione otteniamo il tempo minimo di distribuzione per una rete P2P:

$$D_{P2P} \geq \max \left\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\}$$

### Confronto

Per un confronto diretto tra le due architetture, poniamo  $F/u = 1$  ora,  $u_s = 10u$  e  $d_{min} \geq u_s$ , ovvero abbiamo posto che un peer può trasmettere l'intero file in un'ora, la frequenza di trasmissione del server è 10 volte quella del peer e (per semplificare) le frequenze di trasmissione dei peer sono grandi abbastanza da essere irrilevanti. Otteniamo la seguente figura.

Si nota subito che per l'architettura client-server il tempo di distribuzione risulta essere lineare come calcolato, ma ancora più notevole è il tempo di distribuzione per l'architettura P2P che non solo è sempre minore della controparte client-server, ma addirittura risulta quasi costante all'aumentare del numero dei peer. Questo dimostra come le reti P2P siano estremamente scalabili, e non c'è da stupirsi come il successo di una particolare rete dipenda essenzialmente dal numero di peer presenti. %FIXME: questa è detta molto male.



---

### 1.1.2 Ricerca di informazioni

Si è parlato all'inizio della sezione di come i peer siano consapevoli di cosa altri peer stanno scaricando. Questo sottintende che esiste un sistema all'interno della rete per stabilire quanti peer sono connessi e cosa stanno condividendo e/o scaricando. Ciò significa che tale meccanismo si deve attivare ogniqualvolta un peer si (dis)connette (d)alla rete e un file viene rimosso/aggiunto. L'operazione di aggiunta/rimozione dinamica di un peer/file prende il nome di **churn**. Vediamo quindi come è possibile ricercare informazioni all'interno di una rete P2P quindi, nello specifico del file-sharing, vediamo come sia possibile per un peer sapere quali file sono a disposizione presso quali altri peer.

#### Directory centralizzata

La prima, grande applicazione P2P commercializzata (*Napster*) faceva uso di un indice centralizzato localizzato su un potente server centrale. Quando un utente si collegava alla rete, l'applicazione Napster informava tale server dell'indirizzo IP della macchina e del nome dei file che l'utente ha scelto di condividere con la rete. Il server raccoglie quindi queste informazioni da tutti peer collegati e aggiorna il proprio indice collegato ciascun nome di file agli indirizzi IP dei peer che in quel momento lo stanno condividendo. Ovviamente alla disconnessione di un peer l'indice sarà aggiornato di conseguenza.

Si noti come questa sia di fatto un'architettura ibrida: client-server nella ricerca dei contenuti, P2P nel trasferimento degli stessi.

La semplicità di questo metodo nasconde però alcuni gravi difetti. Il server dell'indice si trova infatti ad essere l'anello debole della rete: un suo guasto ha come conseguenza il crollo dell'intera rete, impedendo di fatto le ricerche. Inoltre tale server deve potenzialmente gestire milioni di utenti contemporaneamente (in quanto si è visto che la potenza delle reti P2P sta nel numero di peer) e quindi risulta essere estremamente costoso da installare e mantenere, oltre a rappresentare un potenziale collo di bottiglia per l'intera rete (Napster era infatti nota per i gravi problemi di traffico da cui era afflitta).

Esiste inoltre un terzo problema non strettamente tecnico ma fondamentale per l'argomento di questo documento: **la privacy dell'utente**. Con un indice centrale che associa un file ad una serie di IP, è facile per chiunque vedere chi possiede cosa, soprattutto nel caso in cui i file condivisi rappresentino una violazione del diritto d'autore. Pur essendo il caricamento dei file una responsabilità dei singoli utenti e non di chi fornisce il servizio, molte legislazioni internazionali possiedono leggi per la tutela del copyright che possono costringere il proprietario del server ad interrompere il servizio, bloccando così l'intera rete, anche quei peer che condividevano materiale lecito. Questo controllo a volte molto invasivo (se non dannoso) da parte di terze parti non coinvolte nella rete è uno dei motivi che ha portato alla nascita di reti sempre più decentralizzate, come appunto la rete Bitcoin.

---

## Query flooding

Rimanendo nell'ambito della ricerca di informazioni per file-sharing, la soluzione diametralmente opposta alla precedente risiede nell'approccio completamente distribuito del query flooding, implementato originariamente dal protocollo **Gnutella**. Questo approccio prevede che l'indice sia distribuito all'interno della comunità, in particolare ogni peer indicizza solamente quello che intende condividere con gli altri peer.

Nel query flooding viene quindi a formarsi una rete astratta e virtuale che collega i vari peer, una rete di copertura (*overlay network*) in cui i collegamenti tra i nodi non sono di tipo fisico. %fixme: questo è vero anche per napster in effetti..

Il protocollo Gnutella prevede che ogni peer sia collegato al massimo a dieci altri nodi dell'overlay network, nodi che vengono denominati *vicinato*. Se un peer vuole cercare un file, invia messaggi di ricerca ai suoi vicini, i quali instradano tali messaggi ai loro vicini che ripetono l'operazione e così via. Questo è il processo del *query flooding* in cui la rete viene "sommersa" (*flood*) di richieste (*query*). Quando un peer riceve una query, controlla se nel suo indice è presente una qualche corrispondenza e, in caso affermativo, invia al peer che ha effettuato la richiesta un messaggio di successo (*query-hit*) contenente il nome del file e la dimensione. La query-hit viene inviata al nodo che ha effettuato la ricerca seguendo lo stesso percorso seguito dal pacchetto query. Dopo qualche tempo il peer di origine avrà un elenco di peer che condividono file corrispondenti alla sua richiesta.

Ma anche in questo approccio, la semplicità nasconde alcune problematiche. La più grave riguarda le ricerche, che vengono propagate all'intera rete di copertura generando una enorme quantità di traffico nella rete sottostante (nel caso di Gnutella, Internet) che connette i peer. Come soluzione a tale problema si è inserito nei messaggi di ricerca di Gnutella un campo *time-to-live*, il quale viene decrementato da ogni peer prima di reinoltrare la richiesta: quando il campo arriva a 0 la richiesta non viene più inoltrata, limitando quindi il raggio di azione della query. Tuttavia in questo modo si riduce anche il numero di peer contattati e la probabilità di ricevere una query-hit.

Esiste inoltre una difficoltà intrinseca alle reti P2P che Napster aveva avitato: il churm. Il protocollo Gnutella tenta di risolvere il problema in questo modo:

1. Per prima cosa, un peer, chiamiamolo X, che vuole connettersi alla rete deve conoscere almeno un altro peer che è già connesso alla rete (**problema del bootstrap**). I due approcci più comuni consistono nel mantenere in ogni peer una lista di client noti a cui tentare di connettersi e/o, in mancanza di connessione a tali nodi, nello scaricare una lista di nodi attualmente online da un sito *tracker*.
2. Dopo aver ottenuto i nodi di bootstrap, X deve tentare di connettersi ad ognuno di questi nodi, fino a riuscirci con uno che chiameremo Y.

- 
3. Dopo la connessione, X invia ad Y un messaggio di *ping* contenente un campo contatore di peer. Y inoltra tale messaggio a tutte i nodi nella sua rete di copertura, i quali continuano l'inoltro fino a quando il contatore non si azzerà.
  4. Ogni volta che un peer Z riceve il *ping*, risponde inviando un messaggio *pong* contenente il proprio IP attraverso la rete di copertura fino ad X.
  5. Grazie ai messaggi *pong*, X viene a conoscenza di molti altri peer online (quanti dipende dal contatore impostato nel messaggio *ping*) e può tentare di connettersi ad essi per ampliare il suo vicinato.

### Copertura gerarchica

Avendo descritto gli approcci diametralmente opposti, vediamo ora la proverbiale via di mezzo, il modello di copertura gerarchica. L'obiettivo è ottenere il meglio dalle due implementazioni sopra descritte, e fu per la prima volta realizzato da FastTrack, un protocollo implementato in Kazaa e Morpheus. Una variante di questo modello è tutt'oggi utilizzata dall'evoluzione di Gnutella, Gnutella2. Come per il query flooding, anche questo è un modello decentralizzato, ma a differenza di tale modello (e a discapito del principio Peer-to-Peer) non tutti i peer sono uguali: come il nome fa suggerire, esiste una gerarchia di peer che assegna a quelli più potenti (leggasi, quelli con più banda) alcune responsabilità in più designandoli come leader per altri peer. Ogni nuovo peer deve stabilire una connessione con un leader a cui notifica tutti i file che intende condividere, creando quindi una sorta di mini indice centralizzato rispettivamente ai peer connessi a quel leader (solitamente nell'ordine del centinaio). A differenza del modello centralizzato però, i leader non sono server dedicati, bensì peer veri e propri in collegamento tra di loro. Il risultato è che i leader sono collegati ad una rete di copertura del tutto simile a quella del modello query flooding, modello utilizzato infatti per l'inoltro delle ricerche. La ricerca di un file da parte di un peer segue quindi due fasi: una richiesta al proprio leader che provvede a consultare il suo indice e una eventuale propagazione della richiesta tramite query flooding da parte del leader agli altri leader. Questa struttura "a strati" permette quindi la connessione di molti peer senza generare una quantità eccessiva di traffico nella rete "ospite".

### DHT (Distributed Hash Table)

Crea un indice completamente distribuito che fa corrispondere gli identificatori dei file alla loro posizione. Consente agli utenti di determinare tutte le posizioni di un file senza generare un'eccessiva quantità di traffico di ricerca. Overnet (Kademlia) sfrutta DHT come anche BitTorrent.



## Capitolo 2

# Bitcoin: moneta elettronica decentralizzata

Tutte le reti P2P finora descritte sono in circolazione da molti anni, hanno una base di utenti che conta milioni di peer divisi tra utenti reali e server automatizzati, contano migliaia di forum di supporto e scambiano quotidianamente una immensa fetta del traffico totale della rete Internet (tanto che molti provider tendono a limitarne quanto più possibile l'utilizzo, soprattutto nelle fasce orarie di maggior traffico). Sono però tutte reti dedicate al file-sharing. Bitcoin no. O almeno, non proprio, come vedremo.

Bitcoin è una rete P2P (intesa per tutti e tre i livelli descritti in precedenza) che mira a creare un sistema di valuta digitale privo di controllo centrale, con pagamenti effettuati direttamente tra gli utenti senza l'intervento di terzi. È stata ideata e realizzata in origine da un anonimo noto con il nome di **Satoshi Nakamoto** [?], formalizzata con un whitepaper nel 2008 ed entrata in vigore il 3 Gennaio 2009 e si è in poco tempo evoluta in modo esponenziale fino a catturare di recente l'attenzione dei media internazionali, delle banche mondiali e, per alcuni suoi utilizzi illeciti, da FBI ed NSA. Ma vediamo di cosa si tratta.

### 2.1 Moneta Elettronica

**Bitcoin** è il nome dato alla rete, al client originale, al protocollo di comunicazione e alla moneta utilizzata per le transazioni all'interno della rete.

Le monete (d'ora in avanti **btc**) posso essere ottenute “gratuitamente” dopo aver impegnato la propria CPU o GPU in alcuni calcoli di crittografia (operazione chiamata **mining** e discussa più avanti), oppure acquistate da altri utenti della rete tramite una valuta reale <sup>1</sup>.

Entrambi questi metodi sono fondamentali nell'ecosistema Bitcoin:

---

<sup>1</sup>Ad esempio tramite il sito Internet Mt. Gox [?].

- 
- Il mining è una diretta conseguenza della partecipazione di un nodo alla rete (vedi più avanti NETWORK) ed è l'unico metodo con cui vengono create e messe in circolazione nuove **btc**. Per ogni operazione di mining avvenuta con successo si riceve una quantità fissa di btc che viene dimezzata nel tempo, limitando il numero massimo di bitcoin in circolazione a circa 21 milioni. A Gennaio 2013 è stata generata circa la metà delle monete totali e si stima di arrivare a 3/4 nel 2017. La quota massima di moneta circolante e l'assenza di istituti centrali in grado di creare nuove monete rendono l'economia bitcoin invulnerabile all'inflazione che colpisce le economie reali. Il sistema è ispirato a quella che era l'economia del Dollaro prima della istituzionalizzazione della Federal Reserve come banca federale: il valore del Dollaro era legato al valore corrente dell'oro, il quale esiste in quantità limitata ed è ottenibile solo attraverso il lavoro dei minatori.
  - La compravendita di bitcoin è invece simile alle compravendite di azioni effettuate nelle borse di tutto il mondo. Esistono infatti alcuni luoghi dedicati (ad esempio il sito internet *Mt. GOX*) che fungono da stock exchange offrendo agli utenti la possibilità di mettere in vendita o di acquistare bitcoin al prezzo che preferiscono. Sono delle vere e proprie borse che trattano unicamente bitcoin invece che molti titoli di aziende diverse, calcolano un valore di scambio medio basato sulle ultime transazioni portate a termine ma lasciano libero l'utente di scegliere a quanto vendere o comprare bitcoin, con prezzo medio che si adegua di conseguenza.

Come per le monete in valuta reale e le azioni borsistiche, anche le bitcoin vengono “tenute” in portafogli. Come vedremo, il termine *tenute* è usato impropriamente, ma questa è l'apparenza dal punto di vista dell'utente, per cui a tale apparenza al momento ci atterremo.

## 2.2 Portafogli e indirizzi

La prima volta che un nuovo utente avvia il suo client Bitcoin fresco di installazione, si vede assegnato un portafoglio contenente un indirizzo e una copia di chiavi di cifratura simmetrica. L'indirizzo è semplicemente una stringa di 27-34 caratteri alfanumerici che inizia con un 1 o con un 3, generata in modo casuale dalle chiavi create per l'utente. Gli indirizzi rappresentano il punto di uscita e/o il punto di ingresso per tutti i movimenti che coinvolgono bitcoin. Questo significa che nelle transazioni bitcoin compariranno unicamente questi indirizzi, rendendo di fatto **anonimi** tutti i movimenti di bitcoin (meno quelli che riguardano l'acquisto di bitcoin tramite moneta reale), in un modo del tutto equivalente a quello dei conti in Svizzera. Non esiste quindi nessuna correlazione diretta e ovvia tra un utente e il suo indirizzo. Un portafoglio può contenere più indirizzi: basta infatti generare una nuova coppia di chiavi e verrà generato anche un nuovo indirizzo. Il numero di indirizzi esistenti è virtualmente infinito.

---

I portafogli sono solitamente legati al software che li crea, basta quindi cambiare software per poter creare un nuovo portafogli e una nuova coppia di chiavi (oppure installare un software in grado di gestire più portafogli). In alternativa è possibile “aprire un conto” presso numeri siti che offrono questa funzionalità, quale ad esempio blockchain.info. In questo caso però bisogna fare i conti con la sicurezza del sito in questione

## 2.3 Transazioni

Le transazioni rappresentano il nucleo fondamentale di bitcoin. Esse sono il metodo con cui, a livello astratto, le bitcoin vengono trasferite da un account all'altro e tramite la loro analisi ci si assicura che un indirizzo contenga esattamente quel numero di bitcoin, che una bitcoin non venga spesa più volte e che quella bitcoin appartiene a quello specifico indirizzo. Le transazioni si basano su meccanismi di crittografia a chiave pubblica, rendendo quindi obsoleto il coinvolgimento di terze parti nella transazione. Se infatti nelle normali compravendite online, volenti o nolenti si è costretti a fidarsi di terze parti che garantiscono per il buon esito dell'operazione (istituti di credito, compagnie di carte di credito, siti come Paypal, ecc), qui gli utenti hanno direttamente la prova crittografica senza aver quindi necessita di fidarsi di qualcuno.

Satoshi Nakamoto descrive la sua moneta elettronica come una serie di firme digitali. Il trasferimento di moneta da un utente all'altro avviene infatti applicando la firma digitale dell'acquirente ad un hash di una precedente transazione e della chiave pubblica del venditore, e aggiungendo ciò alla fine della moneta.

Riassumendo schematicamente:

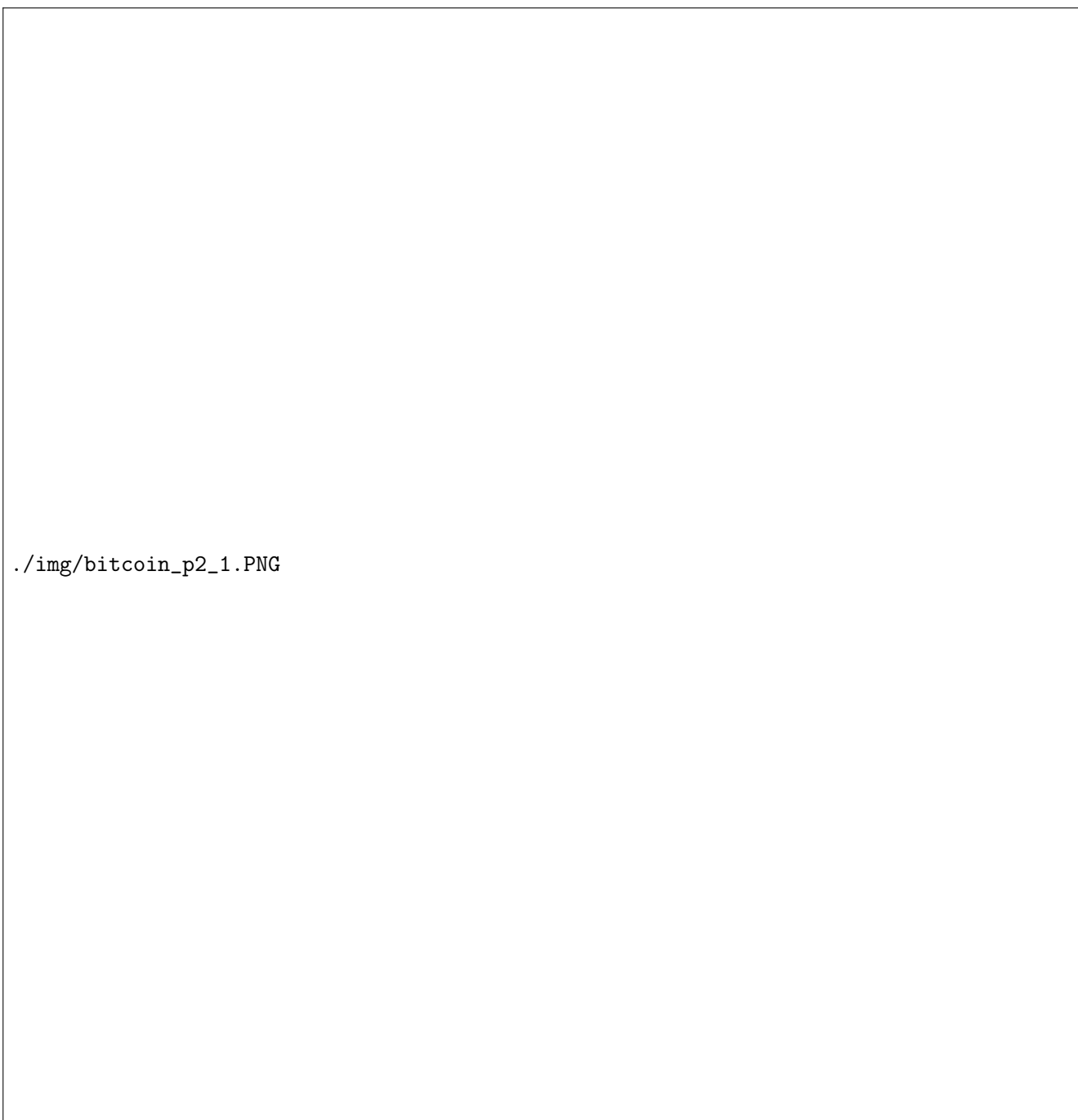
```
hash = hash(previous_transaction, vendor_public_key)
transaction = sign(hash, private_key)
```

La moneta diventa quindi non una unità atomica, ma il risultato di una serie di transazioni che coinvolgono firme digitali e verifiche che deve essere calcolato dinamicamente. La serie di tutte le transazioni mai effettuate viene raccolta in una sequenza denominata **blockchain**.

Questa implementazione però non garantisce che l'acquirente non abbia già effettuato una transazione con questa moneta, ovvero che stia spendendo una moneta già spesa in precedenza.

L'unico modo per garantire ciò senza utilizzare una terza parte di cui fidarsi, è tenere conto di **tutte** le transazioni. Questo vuol dire che tutte le transazioni devono essere annunciate ad un pubblico in grado di mettersi d'accordo sull'effettivo ordine temporale in cui sono state effettuate. Il venditore deve avere quindi la prova che, nel momento in cui riceve la transazione, la maggioranza dei nodi è d'accordo che quella è la prima transazione ricevuta.

---



./img/bitcoin\_p2\_1.PNG

Figura 2.1: Utilizzo delle chiavi e degli hash in una sequenza di transazioni.



---

## 2.4 Timestamp e Proof-of-Work

La soluzione consiste nell'utilizzo di un **timestamp server**. Un timestamp server funziona calcolando l'hash di un blocco di oggetti di cui si vuole realizzare il timestamp e rendendo tale hash pubblico. Il timestamp dimostra inequivocabilmente che gli oggetti esistevano al momento dell'hashing. Ogni timestamp include anche il precedente timestamp nell'hash, formando quindi una catena in cui ogni timestamp rinforza <sup>2</sup> quelli precedenti.

Ora il problema consiste nell'implementare questo server di timestamp in modo distribuito, come è appunto la rete Bitcoin. Per prima cosa bisogna trovare un sistema per cui effettuare il timestamp è un'operazione difficoltosa (computazionalmente parlando), ma verificare che il timestamp sia corretto deve essere immediato. Basandosi sul lavoro di Adam Back ([?]), Nakamoto ha deciso che la difficoltà dell'operazione deve essere trovare un valore che, una volta sottoposto ad hashing (ad esempio con SHA-256), il risultato sia un hash che comincia con uno specifico numero di bit pari a zero: la difficoltà del lavoro è esponenziale al numero di bit zero richiesti, ma è facilmente verificabile con un singolo hash. L'implementazione per bitcoin consiste quindi nella creazione di un blocco di dati di cui calcolare l'hash che contiene le transazioni interessate, l'hash precedente e un valore chiamato **nonce** da incrementare fino a quando l'hash non avrà le caratteristiche richieste. Modificare una transazione comporta modificare un blocco, e quindi ripetere tutto il lavoro di calcolo della nonce. Inoltre, se a questo blocco è già stato incatenato uno più blocchi successivi, anche tali blocchi andranno ricalcolati in sequenza, rendendo il lavoro estremamente gravoso.

Con la prova di lavoro si risolve anche il problema di cosa significa che la maggioranza deve accettare un timestamp. Con l'hash infatti si realizza una sorta di sistema one-CPU-one-vote, e la "decisione della maggioranza" è rappresentata dalla più lunga sequenza di timestamp, che è la sequenza per la quale è stata impiegata la maggior parte di lavoro computazionale. Ciò significa che se la maggior parte della forza-CPU è controllata da peer onesti (cioè che non hanno nessuna intenzione di modificare una transazione effettuata), un nodo disonesto che volesse modificare una transazione non solo dovrebbe rifare tutti i calcoli per il blocco della transazione e per tutti i blocchi successivi, ma avendo minor potenza di CPU a disposizione rispetto ai nodi onesti, verrebbe rapidamente soverchiato dal numero di calcoli da fare, in quando il numero di blocchi da ricalcolare sarebbe sempre superiori a quelli da lui già ricalcolati.


Si capisce subito che è nella rete bitcoin (e nelle reti P2P in generale) è importante che le risorse (potenza di calcolo in questo caso) siano equamente distribuite tra i peer, in modo da evitare che un solo nodo o un solo gruppo di nodi controlli l'intera rete.

Per far fronte alle differenti configurazioni hardware degli utenti, alla sempre crescente capacità di calcolo di CPU e GPU e anche ai potenzialmente mutevoli

---

<sup>2</sup>leggasi: rende più difficili da modificare.

---



`./img/bitcoin_p3_1.PNG`

Figura 2.2: Struttura minimale di una sequenza di blocchi.

---

interessi dei nodi, la difficoltà della prova di lavoro (ovvero il numero di bit zero) è determinata da una media calcolata sul numero medio di blocchi generati ogni ora. Se vengono generati troppi blocchi, vuol dire che la difficoltà è troppo bassa e viene subito aumentata.

## 2.5 Network

A questo punto abbiamo una prima approssimazione di come funziona la rete bitcoin:

1. Le nuove transazioni sono inviate a tutti i nodi.
2. Ogni nodo raccoglie le transazioni che riceve in un blocco.
3. Per ogni blocco, ogni nodo cerca di calcolare una proof-of-work.
4. Una volta trovata la prova, invia il blocco a tutti i nodi.
5. I nodi accettano il nuovo blocco se e solo se tutte le transazioni in esso sono valide (si verifica calcolando l'hash delle transazioni e confrontandole con l'ultimo blocco accettato) e non già spese in precedenza.
6. Il nodo esprime la sua accettazione del blocco appena arrivato mettendosi al lavoro per crearne uno nuovo, usando l'hash del nodo accettato.

I nodi considerano la catena più lunga quella corretta (e viene definita *blockchain*) e lavoreranno sempre in modo da prolungarla. Esiste la possibilità che uno stesso nodo riceva due versioni diverse dello stesso blocco in contemporanea. In questo caso, lavoreranno sul primo blocco ricevuto, ma manterranno una copia anche dell'altro nel caso in cui si rivelasse appartenente alla catena più lunga. La verifica viene fatta non appena viene trovata la nuova proof-of-work e una delle due catene si allunga: a questo punto si individua il blocco da mantenere in base all'hash contenuto nel blocco appena arrivato, gli altri blocchi vengono scartati e si continua il procedimento. Inserendo un po' di terminologia, chiamiamo il primo blocco mai realizzato (che è codificato all'interno di ogni client Bitcoin) *blocco genesi*, l'ultimo blocco della catena *blocco di testa*, la distanza tra un blocco  $b$  e il blocco genesi viene definita *altezza*.

Quando si inviano in broadcast le nuove transazioni, non è necessario che esse raggiungano tutti i nodi: fintanto che raggiungono quanti più nodi possibile, verranno velocemente inglobate in un blocco. I blocchi invece devono essere ricevuti da tutti i nodi, per questo se un nodo riceve un blocco e si accorge (tramite hash) che il blocco precedente gli manca, ne richiederà immediatamente una copia ad un altro nodo, e ripeterà la verifica fino ad ottenere la catena integrale.

Per convenzione, la prima transazione di un blocco è una transazione speciale che crea una nuova moneta e la assegna al creatore del blocco. In pratica il primo nodo che riesce a trovare la proof-work di un nuovo blocco riceve un premio in bitcoin. Tale premio serve ad incentivare i nodi a mantenere attiva la rete ed inoltre permette la messa in circolazione di nuove monete, senza che sia necessaria una zecca centrale. Questo profitto derivante dal calcolo della proof-of-work viene denominato **mining**

---

e ha spinto molti utenti ad acquistare hardware dedicato sempre più performante in modo da creare per primi il nuovo blocco e ottenere il relativo premio, inizialmente ammontante a 50btc ma ora dimezzato a 25btc).

Visto che il numero massimo di btc è stato determinato a priori ed è invariabile, per incentivare i nodi anche quando il mining risulterà inutile, sono state introdotte delle vere e proprie tasse di transazione, le **transaction fees**. Una transaction fee non ha un valore fisso, ma viene decisa da chi effettua la transazione e può anche essere nulla: viene registrata come una differenza tra il valore di input e il valore di output della transazione, con quest'ultimo valore inferiore del primo (come in una tassa, l'acquirente spende più dell'importo effettivo). Tale differenza verrà trasferita nella transazione dedicata all'incentivo al momento della creazione del nuovo blocco, sempre ammesso che l'utente che ha creato il blocco voglia ricevere queste btc.

Oltre ad invogliare un nodo a rimanere attivo, gli incentivi incoraggiano i nodi a rimanere onesti: infatti se un attaccante avido riuscisse ad accumulare abbastanza potenza di calcolo da surclassare quella di tutti gli altri nodi, potrebbe scegliere se truffare gli altri nodi ritirando i suoi pagamenti precedenti oppure usarla per accumulare nuove monete con gli incentivi. Un attaccante si vede quindi incoraggiato a mettere la sua potenza di calcolo a favore del sistema facendogli guadagnare più btc di tutti gli altri nodi messi insieme, invece di usare la stessa potenza per minare le basi dello stesso sistema in cui egli stessi investe i propri soldi.


## 2.6 Risorse necessarie

Il sito Blockchain.info [?] offre vari servizi agli utenti bitcoin, tra i quali spiccano un portafogli online, un sistema di navigazione dell'intera catena dei blocchi e dettagliate statistiche sull'intera rete. Da tale sito si vede come in media, in 24 ore vengono effettuate 56700 transazioni archiviate in 220 blocchi, il che vuol dire un blocco ogni 6.55 minuti. Se ogni nodo dovesse mantenere ogni singola transazione, lo spazio di memoria occupato renderebbe la rete bitcoin non così appetibile per l'utente medio. Risulta necessario minimizzare la quantità di memoria necessaria a mantenere la blockchain senza compromettere la sicurezza.

Si è detto infatti che l'hash di un blocco viene calcolato a partire dall'hash del blocco precedente, da una nonce e dalle transazioni contenute nel blocco. Ma invece che memorizzare le intere transazioni, esse vengono inserite come foglie di un albero di Merkle [?], una struttura dati in cui ogni nodo non foglia è l'hash di tutti i suoi figli, in modo che nella radice di tale albero ci sia un solo hash che riassume in se tutte le transazioni. È pertanto possibile calcolare l'hash di un blocco basandosi unicamente sull'hash del blocco precedente, sulla nonce e sulla radice dell'albero di Merkle, ovvero sull'hash riassuntivo delle transazioni, rendendo quindi possibile eliminare alcune transazioni dalla blockchain per risparmiare un poco di spazio.

Con questa struttura, il **block header** viene ad occupare esattamente 80 Bytes (vedi sezione tecnica per i dettagli), il che significa circa 6.2 MB all'anno (Satoshi Nakamoto con una stima di un blocco ogni 10 minuti aveva previsto 4.2 MB annui).

---




./img/bitcoin\_p4\_1.PNG

Figura 2.3: Struttura di un blocco prima e dopo la compressione dell'albero delle transazioni.

---

Una quantità decisamente ridotta che rende la blockchain tollerabile su ogni computer degli ultimi 7 anni.

È possibile quindi verificare i pagamenti senza disporre di un nodo personale. L'utente deve possedere una copia degli header della catena più lunga (che può ottenere dai nodi della rete) e ottenere il ramo di Merkle che collega la transazione al blocco in cui è stata inserita. Non può verificare la transazione da solo calcolando gli hash (non ha le altre foglie dell'albero di Merkle), ma collegandola ad un punto specifico della catena può vedere che un nodo l'ha accettata, ed eventuali blocchi successivi confermano che anche l'intera rete l'ha accettata.



./img/bitcoin\_p5\_1.PNG

Figura 2.4: Struttura di un blocco prima e dopo la compressione dell'albero delle transazioni.

Così come è descritta, la verifica è affidabile fin tanto che la rete è controllata da nodi onesti, ma è vulnerabile se la rete è soverchiata da un attaccante. Mentre un nodo può effettuare le verifiche da se, il metodo semplificato è vulnerabile alle transazioni ad-hoc create da un attaccante che controlla la rete. Una strategia di

---

difesa consiste nell'accettare avvisi dai nodi della rete quando questi rilevano blocchi non validi, richiedendo all'utente di scaricare l'intero blocco. Per questo motivo chi utilizza bitcoin per business ritiene preferibile avere un nodo personale con intera blockchain, in modo da poter effettuare verifiche autonomamente e velocemente.

Da questo si può dedurre un fatto importante che distingue la rete bitcoin dalle altre reti P2P: si può partecipare alla rete anche senza il relativo software (in questo senso, si partecipa al livello comunitario della rete): non serve essere un nodo, basta essere un utente.

## 2.7 Gestione dei valori

Anche se è possibile trattare le monete una ad una, è improponibile fare una transazione per ogni singolo centesimo. Per la divisione degli importi, le transazioni contengono molteplici input e output. Nella maggior parte delle transazioni da indirizzo ad indirizzo si avranno o un singolo input proveniente da una grande transazione precedente oppure più input provenienti da più transazioni piccole precedenti, e al massimo due output: uno per il pagamento vero e proprio e uno per restituire il resto, se presente, al mittente. È importante distinguere input e output dal pagamento effettuato ad un indirizzo: una transazione è infatti l'operazione tramite il quale uno ed un solo utente assegna una certa quantità di BTC prelevate da uno o più dei suoi indirizzi ad uno o più indirizzi anche non appartenenti allo stesso utente<sup>3</sup>. Per cui per ogni indirizzo a cui si desidera assegnare BTC si creerà una voce di output contenente l'indirizzo e il valore da assegnarli, ed inoltre se gli input e gli output differiscono verrà creato un ulteriore output per inviare il resto al creatore della transazione su un nuovo indirizzo appositamente creato.

Riassumendo:

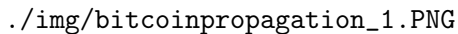
- Un input è un riferimento ad un output di una precedente transazione che contiene l'indirizzo di chi sta effettuando la transazione. Se ci sono più input, l'importo degli output da loro referenziati viene sommato ed il totale è il massimo valore utilizzabile dall'output.
- Un output contiene l'indirizzo del destinatario e il valore da spedire. Dato che, in una futura transazione, un output può essere referenziato da un solo input, potrebbe verificarsi il caso in cui l'input sia maggiore dell'output e della transaction fee desiderata. In questo caso è necessario creare due output, uno con il valore da spedire e l'indirizzo del destinatario, l'altro con la differenza da restituire al mittente, il **resto**. La differenza tra il totale degli input e il totale degli output è la transaction fee.

La situazione è visibile in 2.5.

---

<sup>3</sup>Senza analizzare la rete come descritto in 3 è infatti impossibile risalire al proprietario di un indirizzo.

---



./img/bitcoinpropagation\_1.PNG

Figura 2.5: Schema di una transazione in cui vengono evidenziati input e output.

Nonostante la stretta dipendenza tra le varie transazioni, non è necessario estrarre l'intero background di ogni input, in quanto la transazione verrà accettata solo se il blocco che contiene le transazioni con gli output referenziati dagli input è stato accettato dalla rete.

Dato il valore elevato di una singola btc (che può variare da poche decine a centinaia di dollari) e le difficoltà inerenti nel trattare numeri in virgola mobile su un computer, l'unità di misura base della transazione non è il btc ma il Satoshi <sup>4</sup>, e  $1 \text{ BTC} = 100000000 \text{ Satoshi}$ .

Questo sistema del calcolo del valore di una transazione è lo stesso utilizzato dal software che gestisce il portafoglio per calcolare il proprio valore: per ogni indirizzo all'interno del portafoglio, il software scansiona le transazioni presenti nella blockchain che contengono l'indirizzo in esame, somma i valori entranti nell'indirizzo, sottrae quelli uscenti e ricava il valore "contenuto" nell'indirizzo. Usando questo sistema, i possedimenti di un utente sono temporalmente limitati all'ultimo blocco accettato nella catena, e non è quindi possibile spendere moneta ricevuta da una transazione non ancora approvata <sup>5</sup>.

%%%%%%%%%% % La documentazione di Satoshi prevede anche privacy e calcoli statistici sugli attacchi. % Visto che prevedo una sezione apposta per anonimato e sicurezza, ne parlo la e non qua. % Attenzione che la parte di topologia successiva

---

<sup>4</sup>dal "nome" dell'inventore di Bitcoin

<sup>5</sup>tecnicamente, nessuna moneta è ricevuta fin tanto che la transazione non è approvata. L'utente non è nemmeno consapevole della transazione a lui destinata fino ad avvenuta approvazione.



---

nomina alcuni risultati statistici di questa parte... una volta scritta saranno da fare i collegamenti

## 2.8 Analisi della rete

Sappiamo che i nodi della rete Bitcoin sono tutti omogenei e nessuno di essi ha un ruolo di coordinatore o comunque diverso da quello degli altri nodi, e ognuno di essi mantiene una copia di tutte le informazioni necessarie per far funzionare il sistema. Vediamo ora più formalmente come si struttura la rete Bitcoin e come le informazioni (ovvero, le transazioni e i blocchi) si propagano in essa.

### 2.8.1 Topologia

Non essendoci coordinazione tra i nodi, il grafo rappresentante la rete ha una struttura casuale. Durante il *churn* il nuovo nodo interroga alcuni server DNS gestiti da nodi volontari e si vede restituiti un insieme casuale di nodi con cui fare il bootstrap. Una volta connesso, impara dai suoi vicini gli indirizzi dei nodi raggiungibili e si mette in ascolto nel caso nuovi nodi vengano annunciati in broadcast. A differenza di una rete P2P “tradizionale”, non esiste alcun modo per un nodo di lasciare la rete: gli indirizzi restano memorizzati per diverse ore prima che gli altri nodi li rimuovano dalla loro lista di indirizzi noti.

Ogni nodo tenta di mantenere un certo numero  $p$  di connessioni con gli altri nodi, collegandosi ad un indirizzo scelto a caso tra quelli che conosce nel caso il numero di connessioni sia inferiore a  $p$  ma senza bloccare connessioni in ingresso nel caso tale numero sia superiore. Il valore  $p$  rappresenta quindi un valore minimo di connessioni spesso e volentieri superato per nodi abilitati ad accettare connessioni in ingresso. Per il client *bitcoind*, il primo implementato da Nakamoto e tutt’ora il più diffuso, il default è  $p = 8$ , ma il numero medio di connessioni contemporanee è 32 nel caso in cui non ci siano firewall o NAT <sup>6</sup> ad intercettare connessioni esterne.

Dato il tipo di connessione tra i nodi, le partizioni non sono individuabili nel momento in cui si creano e, nel caso in cui dovessero esistere, esse continuerebbero ad operare indipendentemente. Una tale situazione comporta nel tempo una divergenza tra le situazioni tracciate dalle due partizioni, situazioni potenzialmente incompatibili. Pertanto è fondamentale individuare le partizioni nel momento in cui si creano, e uno dei modi per farlo è tracciare il potere computazionale complessivamente presente nella rete: una rapida diminuzione della frequenza di creazione di blocchi può indicare la presenza di una partizione.

---

<sup>6</sup>Network Address Translator: maschera gli indirizzi di una LAN come un unico indirizzo IP sulla rete Internet, e spesso impedisce a host presenti in Internet di connettersi ad host specifici in una LAN.

---

## 2.8.2 Propagazione delle informazioni

Per ciò che riguarda il mantenimento della blockchain, solamente i messaggi di transazione *tx* e i blocchi sono rilevanti. Tali messaggi sono molto più comuni di tutti gli altri scambiati nella rete e potrebbero raggiungere dimensioni rilevanti. Per evitare di sprecare banda e ridurre l'overhead, è necessario fare in modo di inviare ognuno di questi messaggi una sola volta per ogni nodo, evitando quindi l'inoltro ai nodi che già hanno ricevuto quel messaggio.

L'implementazione utilizzata da Bitcoin sfrutta una tecnica passiva: invece che inoltrare tutti i messaggi, ogni nodo dopo aver verificato un messaggio o una transazione invia a suoi vicini un messaggio *inv* per segnalare la disponibilità di nuovi tx o blocchi. Il messaggio *inv* contiene gli hash dei blocchi e delle transazioni disponibili per l'invio, che ogni vicino può richiedere nel caso gli mancasse tramite un messaggio *getdata*. I messaggi *inv* e *getdata* (61 B) sono ovviamente significativamente più piccoli dei messaggi *tx* e *block* (fino a 500 kB per i blocchi).

Ogni volta che si verifica lo scambio di *inv* e *getdata*, la velocità di propagazione del messaggio subisce un rallentamento, dovuta sia allo scambio di messaggi sia alla necessità di verificare la transazione/blocco con calcoli crittografici: bisogna verificare ogni nodo prima di notificarne la disponibilità, e tale verifica include la verifica di ogni transazione nel blocco, le quali a loro volta richiedono accesso random ai dati del disco rigido.

Vediamo di stimare formalmente come si propaga un dato nella rete.

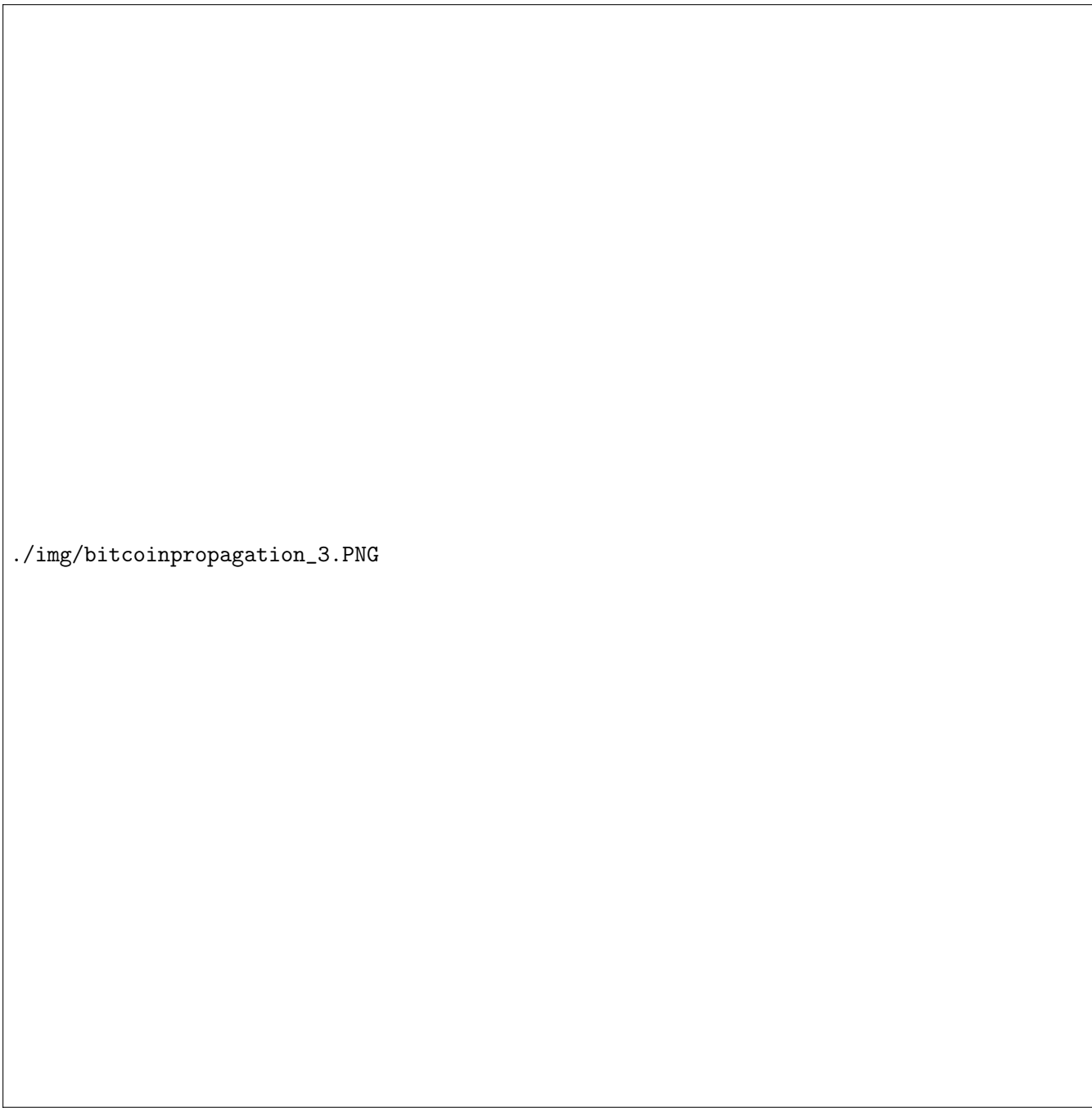
Definiamo  $t_{i,j}$  come il tempo trascorso tra il primo annuncio e il momento in cui il nodo  $j$  riceve l'oggetto  $i$ . Il nodo  $o$  sarà l'origine dell'oggetto  $i$  (ovvero il nodo che ha trovato il blocco o il nodo che ha creato la transazione), quindi  $t_{i,o} = 0$ .

Il tempo  $t_{i,j}$  ha un comportamento **doppio esponenziale**. Il periodo di propagazione segue due fasi: una prima crescita esponenziale in cui la maggior parte dei nodi che ricevono un *inv* richiederanno il relativo dato che non possiedono, e una seconda fase di compressione esponenziale in cui la maggior parte dei nodi che ricevono un *inv* hanno già il dato.

Per effettuare le loro misurazioni, i ricercatori Christian Decker e Roger Wattenhofer dello Swiss Federal Institute of Technology di Zurigo hanno implementato il protocollo bitcoin su un nodo creato in modo tale da non inviare messaggi *inv*, *tx* o *block* e collegato a direttamente a quanti più nodi tradizionali possibile. Tale implementazione ha il solo scopo di tracciare come i blocchi si propagano nella rete restando in ascolto dei messaggi *inv* che ne annunciano la disponibilità [?]. La ricezione di un *inv* implica che il nodo che ha inviato il messaggio ha ricevuto e verificato un blocco.

La rilevazione è partita su nodi di altezza 180000 ed è durata per 10000 blocchi. Le informazioni rilevate includono l'hash del blocco, l'IP del noto annunciante e una timestamp della ricezione dell'*inv*. La stima per  $t_{i,j}$  è ottenuta sottraendo il timestamp del primo annuncio di un blocco da tutti gli annunci successivi per quel blocco. I risultati sono riassunti nel grafico 2.6.

---



`./img/bitcoinpropagation_3.PNG`

Figura 2.6: Istogramma normalizzato del tempo che intercorre dal primo annuncio del blocco con una curva di interpolazione esponenziale.

---

Le misurazioni effettuate ci permettono di stabilire che il tempo mediano in cui un nodo riceve un blocco è di 6.5 secondi, mentre la media è di 12.6 secondi. La lunghezza della seconda fase, quella di contrazione esponenziale, evidenzia come dopo soli 40 secondi il 95% dei nodi abbia ricevuto il blocco.

Abbiamo però detto come la dimensione di un blocco sia variabile, fino a 500kB. I dati precedenti sono aggregati per tutti i blocchi e non tengono conto delle differenti dimensioni degli stessi. Perciò definiamo ora il *costo di ritardo* come il ritardo che ogni kB causa alla diffusione di una transazione o di un blocco. Tale costo risulta essere una combinazione sia del tempo di trasmissione che del tempo di verifica. I risultati sono contenuti nel grafico 2.7.

Per pacchetti di dimensione superiore ai 20kB si vede come il costo sia pressoché costante, mentre per dimensioni minori si assiste a notevole ritardo. La causa di ciò sta nel **ritardo da un roundtrip**, ovvero il fatto che anche i piccoli messaggi vengono annunciati con *inv* e richiesti con *getdata*. Il roundtrip è dominante per le transazioni in quanto il 96% di tutte le transazioni sono inferiori ad 1kB. Per i blocchi, la cui dimensione è per la maggior parte superiore ai 20kB, ogni kB di dimensione costa circa 80ms di ritardo fino al momento in cui la maggioranza dei nodi non ha il blocco. Nel caso di piccoli blocchi sarebbe pertanto ottimale evitare l'annuncio e inoltrare direttamente il blocco ai nodi vicini.

### 2.8.3 Informazioni scomparse

Trattiamo ora il caso in cui un blocco inoltrato in rete porti ad un fork della blockchain che viene rilevato solo da una piccola parte dei nodi.

Definiamo il grafo che descrive la rete come  $G = (V, E)$ , con  $V$  i nodi (vertici) ed  $E$  le connessioni tra i nodi (archi/edges). Definiamo inoltre la partizione  $P_h \subset V$  come l'insieme dei nodi il cui blocco di testa si trova ad altezza  $h$ . Trovare un nuovo nodo  $b_{h+1}$  crea una nuova partizione  $P_{h+1,b}$  contenente i nodi che considerano questo nuovo blocco come blocco di testa, in altre parole questo è il primo blocco di altezza  $h + 1$  che abbiano ricevuto. Se non vengono trovati altri blocchi, allora i nodi di  $P_h$  adiacenti a  $P_{h+1,b}$  si uniscono a  $P_{h+1,b}$  lasciando (e quindi eliminando) la partizione  $P_h$ . Dal'altro canto, se viene trovato un blocco  $b'_{h+1}$  da un nodo in  $P_h$  viene creata una nuova partizione  $P_{h+1,b'}$ . Anche in questo caso i nodi di  $P_h$  abbandoneranno la loro partizione per unirsi ad una delle due nuove di altezza superiore. Solamente i nodi di  $P_h$  che sono a contatto sia con  $P_{h+1,b}$  che con  $P_{h+1,b'}$  saranno consapevoli dell'esistenza di entrambe le partizioni e quindi del fork della blockchain, e considereranno invalido il blocco di altezza  $h + 1$  proveniente dall'altra partizione e di conseguenza non lo annunceranno ai loro vicini fermando quindi l'espansione della partizione. Tale meccanismo si applica anche alle transazioni: se due tx tentano di spendere lo stesso output, solo la prima transazione ricevuta da un nodo verrà considerata valida, la seconda verrà considerata invalida e non annunciata ai vicini. Questo comportamento ha il vantaggio di impedire ad un nodo malevolo di inondare la rete con centinaia di transazioni contraddittorie senza costo addizionale, in termini di transaction fees, per il nodo malevolo. Il rovescio della medaglia è che questo

---

`./img/bitcoinpropagation_4.PNG`

Figura 2.7: Costo del ritardo per il 50°, 75° e 90° percentile. Il grafico è focalizzato sui valori più bassi delle ascisse per poter mostrare il comportamento costante presente dopo i 20kB.

---

sistema di nascondere le informazioni ritenute sbagliate da un nodo permettere l'implementazione di **attacchi double spend** che risultano invisibili ai commercianti. Nel caso delle transazioni, dato che esse non devono per forza propagarsi a tutti i nodi, il meccanismo descritto è ragionevole e protegge la rete da transazioni spam. Nel caso dei blocchi invece, fermarne la propagazione è controproducente: i fork della blockchain, che con tale sistema sono invisibili per la maggior parte dei nodi, sono una cartina tornasole che indica come nella rete ci siano alcune inconsistenze non risolte. Dato che i blocchi valido ma potenzialmente in conflitto non possono essere creati con un frequenza arbitraria come le transazioni, permettere il loro inoltro anche nel caso di conflitto non crea possibilità per un potenziale attacco.

## 2.9 Fork della blockchain

Durante il normale utilizzo della rete potrebbe capitare di essere testimoni di un fork se si ricevono due blocchi conflittuali, ma osservare tutti fork che avvengono è molto difficile a causa della non propagazione dei blocchi in conflitto discussa in precedenza. Se a questo aggiungiamo il fatto che una partizione potrebbe avere dimensione unitaria (un nodo genera un nuovo blocco in conflitto con il blocco di testa di tutti i suoi vicini) è evidente come per poter rilevare tutti i fork bisognerebbe connettersi ad ogni nodo della rete. Ma abbiamo detto che alcuni nodi non sono raggiungibili dall'esterno, per cui possiamo solo stimare il numero di fork che avvengono.

Utilizzando la configurazione descritta nella sezione precedente, sono stati raccolti tutti i blocchi di altezza compresa tra 180000 e 190000. Essendo un grande campione che coinvolge tutti i nodi raggiungibili, è abbastanza probabile che tutti i blocchi generati siano stati propagati fino al nodo spia implementato permettendo di individuare la maggior parte dei fork avvenuti nell'intervallo di rilevazione. Nei 10000 blocchi osservati sono stati identificati 169 fork, il che si traduce in rateo di forking  $r = 1.69\%$ . L'istogramma 2.8 mostra i risultati in modo più dettagliato.

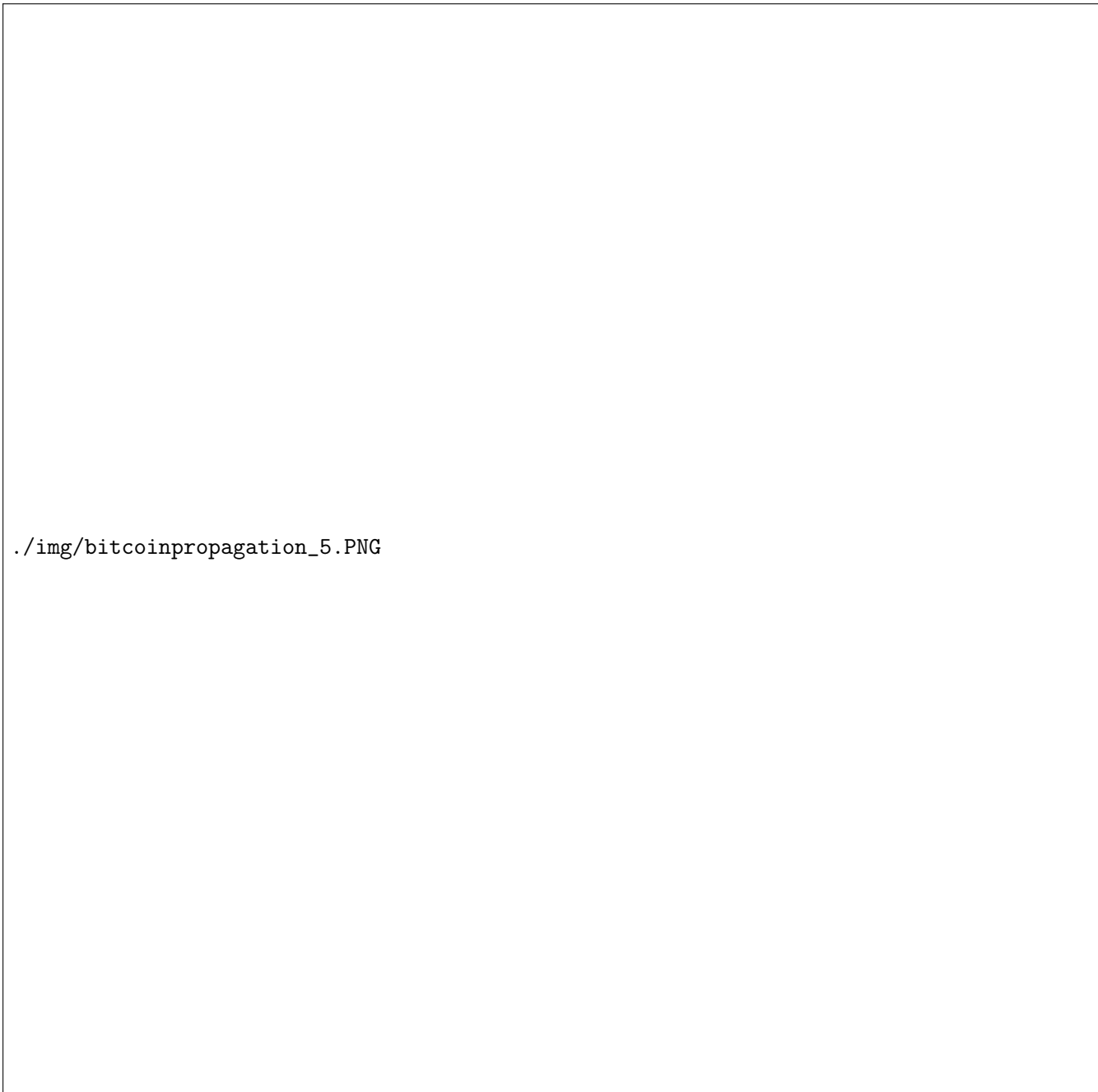
### 2.9.1 Creazione del modello

Il protocollo bitcoin adatta la difficoltà della proof-of-work ogni 10 minuti in modo da mantenerla sufficientemente elevata da essere significativa. Definendo  $X_b$  come la variabile casuale che rappresenta i secondi trascorsi tra il ritrovamento di un nodo e il ritrovamento del nodo precedente, allora la *probabilità che un blocco venga trovato* nella rete in un dato secondo è

$$P_b = \Pr[X_b < t + 1 | X_b \geq t] \approx 1/600$$

Un fork avviene se, durante la propagazione del blocco  $b$ , viene trovato un blocco  $b'$  in conflitto nella parte di rete non ancora a conoscenza di  $b$ . Definiamo  $t_j$  come il tempo in secondi in cui  $j$  apprende dell'esistenza di  $b$  da quando esso è stato creato. La funzione  $I_j(t)$  identifica se il nodo  $j$  sa dell'esistenza di  $b$  nell'istante  $t$ , e

---



./img/bitcoinpropagation\_5.PNG

Figura 2.8: Fork osservati tra i blocchi 180000 e 190000 durante il collegamento alla rete.

---

la funzione  $I(t)$  conta il numero di nodi che hanno ricevuto e verificato  $b$  all'istante  $t$ .

$$I_j(t) = \begin{cases} 0 & \text{se } t_j > t \\ 1 & \text{se } t_j \leq t \end{cases}$$

$$I(t) = \sum_{j \in V} I_j(t) \quad \text{con } V \text{ insieme dei vettori}$$

Da cui ottengo il rateo di nodi informati:

$$f(t) = \mathbb{E}[I(t)] \cdot n^{-1}$$

Notare come la  $f(t)$  sia equivalente alla funzione di distribuzione cumulativa (**CDF**) della frequenza alla quale i peer vengono informati. Possiamo quindi utilizzare la funzione di densità di probabilità (**PDF**) della frequenza con cui i peer vengono informati rappresentata in 2.6 come una approssimazione durante le rilevazioni. Solo i nodi non informati possono produrre blocchi in conflitto, per cui combinando la probabilità di trovare un blocco con il rateo di nodi non informati otteniamo la probabilità di un fork. Definiamo  $F$  come la variabile casuale discreta che conta il numero di blocchi in conflitto trovati mentre un altro blocco viene propagato. Allora la probabilità di un fork risulta:

$$\Pr[F \geq 1] = 1 - (1 - P_b)^{\int_0^\infty (1-f(t)) dt}$$

In questo ultimo passaggio si è assunta la semplificazione per la quale la probabilità di un nodo di trovare un blocco è distribuita uniformemente in modo casuale tra tutti i nodi.

Per cui, sapendo la probabilità dell'intera rete di trovare un blocco  $P_b$  e la distribuzione di come i nodi apprendono dell'esistenza del blocco  $I_j$ , si può derivare la probabilità di un fork. Questi due valori dipendono dal potere computazionale della rete nonché dalla sua topologia e dimensione.

### 2.9.2 Misurazioni

Per confermare il corretto funzionamento del modello proposto è necessario confrontare la probabilità ottenuta con i dati rilevati.

Bisogna innanzitutto notare come i nodi non sincronizzino i loro orologi interni, bensì si regolano sui loro vicini, esiste una differenza non trascurabile tra i timestamp. Ad esempio il blocco ad altezza 209873 ha un timestamp pari a 22:10:13 mentre il blocco ad altezza 209874 ha un timestamp di 22:08:44. Dato che il secondo include l'hash del primo, i blocchi sono stati trovati nell'ordine corretto. Da questo si deduce che il conflitto nei timestamp è derivato dalla mancata sincronizzazione degli orologi dei nodi.

In questa analisi si potrebbe tenere conto dell'orario in cui il nodo spia rileva l'annuncio del blocco e l'orario in cui il blocco è stato trovato. Anche se questo



---

metodo non subisce la differenza di orario tra i nodi, potrebbe esistere un lieve ritardo tra il calcolo del blocco e la rilevazione del relativo annuncio, e per la misurazione abbiamo a disposizione solo il timestamp del blocco. Dato che il calcolo della proof-of-work è un processo di Poisson, la differenza temporale segue una distribuzione esponenziale. La combinazione della differenza temporale degli orologi e il tempo intercorso tra i ritrovamenti dei blocchi causa uno spostamento a destra del massimo. Possiamo correggere la situazione spostando a sinistra fin quando il massimo non risulta in  $t = 0$ . L'orario di annuncio rilevato durante la misura non è influenzato dalla differenza temporale e produce l'istogramma corretto.

$$g(t) = \lambda e^{-\lambda \cdot x}$$

Estraendo i timestamp dai blocchi tra altezza 180000 e 190000 si ottiene la distribuzione illustrata nel grafico 2.9. Interpolando la distribuzione ottenuta con la distribuzione esponenziale si ottiene  $\lambda = 0.001578$  da cui risulta un tempo atteso tra due blocchi pari a  $1/\lambda = 633.68$  secondi. Interpolando la densità di probabilità del tempo tra i primi annunci e le misurazioni si ottiene  $\lambda = 0.001576$  che si traduce in un tempo atteso tra due blocchi  $1/\lambda = 634.17$  secondi. Le due approssimazioni sono coerenti ma sono entrambe leggermente sopra il valore obiettivo di 600 secondi. La differenza è probabilmente dovuta ad un decremento del potere computazione della rete.

Per quando riguarda la propagazione dei nodi nella rete, a causa della normalizzazione il diagramma 2.6 rappresenta anche la funzione di densità di probabilità (**PDF**) delle variabili casuali  $t_{b,j}$  per tutti i blocchi  $b$  dell'intervallo di misurazione. Per cui la frequenza dei nodi informati  $f(t)$  è l'area che sottostà all'istogramma 2.6 fino al tempo  $t$ .

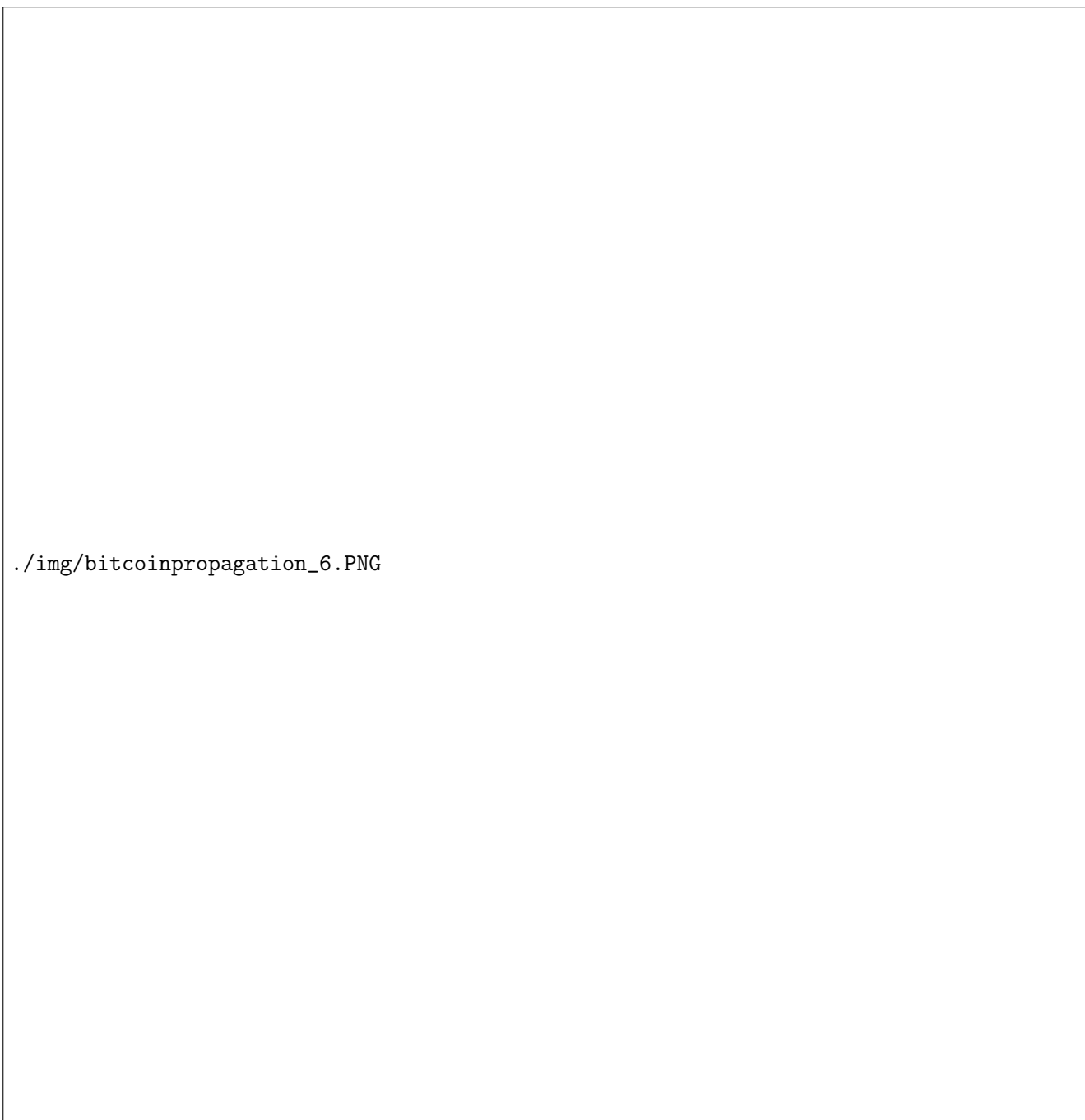
Combinando la probabilità di trovare un blocco e la funzione della frequenza dei nodi informati si ottiene la seguente probabilità per un fork:

$$\begin{aligned} \Pr[F \geq 1] &= 1 - (1 - P_b)^{\int_0^\infty (1-f(t)) dt} \\ &= 1 - \left(1 - \frac{1}{633.68}\right)^{11.37} \\ &\approx 1.78 \end{aligned} \tag{2.1}$$

Comparando questo risultato con quello osservato di 1.69% si nota di aver sovrastimato il valore osservato solo del 5%. Il risultato leggermente superiore può essere spiegato dall'assunto fatto che la potenza di calcolo sia uniformemente distribuita tra tutti i nodi nella rete. Ciononostante, il buon risultato ottenuto dimostra come il modello sia una efficace rappresentazione della realtà.

Dato che il numero di transazioni e le dimensioni della rete molto probabilmente cresceranno mano a mano che aumenta l'adozione di Bitcoin, la frequenza dei fork è destinata a salire. Una rete più grande, con una topologia casuale e il numero di connessioni limitate per singolo nodo, aumenta il suo diametro e la distanza media tra i nodi e l'origine di un blocco. L'aumento del numero di transazioni provoca una

---



`./img/bitcoinpropagation_6.PNG`

Figura 2.9: Tempo di distribuzione spostato a sinistra per i blocchi trovati tra le altezze 180000 e 190000.

---

crescita nella dimensione dei blocchi il che a sua volta aumenta il tempo necessario per la verifica e la trasmissione ad ogni passo della propagazione.

Una interpretazione alternativa del risultato proposto in è che ogni volta che un blocco viene trovato, l'equivalente di 11.37 secondi di potere computazionale dell'intera rete viene sprecato. Infatti il lavoro impiegato per trovare il primo blocco di una blockchain alternativa (che potrebbe essere scartata) non contribuisce alla sicurezza della rete e costituisce un eventuale punto a favore di un attaccante che cerca di implementare una sua blockchain alternativa. Come detto in precedenza da Nakamoto, un attaccante capace di controllare più del 50% del potere computazionale delle rete è in grado di trovare proof-of-work più velocemente di tutto il resto della rete. L'attaccante sarebbe perciò in grado di rimpiazzare l'intera storia delle transazione a partire da un qualsiasi blocco. Pur sicuramente sufficiente, questa condizione non è minima. In realtà l'efficienza della rete come intero, incluso il ritardo di propagazione, non è ottimale. La potenza computazionale effettiva nella rete così come si presenta a Settembre 2013 è pari a  $1 - 11.37/633.68 = 98.20\%$ . Per cui ad un attaccante basta controllare il 49.1% della forza di calcolo della rete per poter portare un attacco e cambiare la blockchain. Al momento questo è un risultato difficile da ottenere, ma la situazione potrebbe cambiare in peggio a causa dell'aumento costante del ritardo di propagazione.



## Capitolo 3

# Privacy

Uno degli obiettivi del design di Bitcoin è l'anonimato dell'utente. Il paragone più appropriato è quello con i conti in banca della Svizzera. Ogni utente bitcoin può infatti possedere più di un portafogli (ovvero di una coppia di chiavi pubblica e privata) e per ogni portafogli più di un conto/indirizzo (che altro non è che una stringa unica generata grazie alla coppia di chiavi) e tutte le transazioni si basano unicamente sull'indirizzo. Non esiste quindi nessun modo per risalire al proprietario di un dato indirizzo bitcoin basandosi unicamente sulla struttura della rete.

Esistono ovviamente diverse tecniche sia per ridurre il livello di privacy, ad esempio tramite il furto delle chiavi o un'analisi statistica degli input e degli output delle transazioni, sia per aumentarlo, magari creando un diverso portafoglio per ogni conto che si desidera creare.

### 3.1 Analisi quantitativa della privacy

Ma come si fa a misurare il livello di privacy offerto da Bitcoin? In questa sezione si discuteranno alcune metriche arbitrarie create appositamente che trattano i vari aspetti dell'anonimato nella rete Bitcoin.

Essendo una analisi quantitativa, è necessario fornire una definizione formale per i termini fin qui usati in modo più o meno descrittivo. Sotto tale ottica verranno quindi proposte alcune definizioni formali, la prima delle quali è il concetto portante della rete:

Transazione:

$$\tau(a_S \rightarrow a_R) = \{\text{source}, B, a_R, \text{SIG}_{\text{sk}_{a_S}}(\text{source}, B, a_R)\}$$

definisce una transazione che intercorre tra i due indirizzi  $a_S$  e  $a_R$ , in cui  $\text{SIG}_{\text{sk}_{a_S}}$  è la firma digitale creata utilizzando la chiave privata  $\text{sk}_{a_S}$  corrispondente alla chiave pubblica associata all'indirizzo  $a_S$ .  $B$  è la quantità di BTC trasferite e source è un riferimento alla più recente transazione da cui  $a_S$  ha ottenuto le  $B$  BTC.

Al fine dell'analisi, assumiamo la situazione tipica della rete Bitcoin, in cui ogni singolo utente dispone di diversi indirizzi a lui associati. Mentre questa può sembrare

---

una forzatura, in realtà non lo è affatto. Come abbiamo detto in precedenza, ogni transazione contiene un indirizzo di ritorno per il resto. Tale indirizzo viene definito *indirizzo ombra* ed è creato automaticamente senza l'intervento dell'utente. Per cui ogni utente possiede almeno due indirizzi: quello creato al momento della creazione del portafogli e l'indirizzo ombra.

### 3.1.1 Il modello antagonistico

Gli algoritmi proposti consistono nell'osservazione del log pubblico di Bitcoin (*pubLog*) durante un periodo  $\Delta t$ . Durante questo periodo,  $n_U$  utenti (dall'insieme  $U = \{u_1, \dots, u_{n_U}\}$ ) contribuiscono al *pubLog* con l'insieme di indirizzi  $A = \{a_1, \dots, a_{n_A}\}$ . Si ha poi l'insieme delle transazioni  $T = \{\tau_1(S_1 \rightarrow R_1), \dots, \tau_{n_T}(S_{n_T} \rightarrow R_{n_T})\}$  in cui  $\tau_i(S_i \rightarrow R_i)$  descrive una singola transazione con ID univoco pari a  $i$ , e con  $S_i$  e  $R_i$  l'insieme degli indirizzi di mittente e destinatario rispettivamente. Viene introdotto anche un avversario  $\mathcal{A}$  interessato ad ottenere informazioni su tutte le transazioni e gli indirizzi appartenenti ad un insieme di utenti Bitcoin. Si assume quindi che  $\mathcal{A}$  sia un nodo legittimo della rete, abbia accesso a *pubLog*, ad alcuni indirizzi di commercianti resi pubblici e altre informazioni statistiche pubblicamente disponibili. Inoltre si assume anche che gli utenti siano incoraggiati a cambiare frequentemente i loro indirizzi, spostando le loro monete da un indirizzo all'altro. Questa è una delle abitudini consigliate da Nakamoto.

### 3.1.2 Quantificazione della Privacy

Esistono (almeno) due nozioni distinte di privacy per la rete Bitcoin.

*Activity unlinkability*: Un avversario  $\mathcal{A}$  non dovrebbe essere in grado di associare due indirizzi distinti (*address unlinkability*) o transazioni (*transaction unlinkability*) ad un utente scelto dell'avversario stesso.

*Profile indistinguishability*: L'avversario non deve essere in grado di ricostruire i profili (insieme di indirizzi e transazioni) di tutti gli utenti del *pubLog*. Questa nozione di privacy è più forte della precedente, in quanto riguarda l'intera rete e non solo un utente. Inoltre i due profili (per transazioni e per indirizzi) non sono equivalenti quando si tratta di modellare il profilo di un utente, in quanto è possibile indovinare correttamente gli indirizzi di utenti coinvolti in poche transazioni ma sbagliare nel caso di pochi indirizzi coinvolti in molte transazioni.

Vengono definiti due algoritmi *AddUnl* e *ProfInd* che implementano una sorta di sfida in cui l'attaccante tenta di violare le due nozioni di privacy sopra descritte. Il risultato sarà una quantificazione delle nozioni di privacy nei termini del vantaggio che  $\mathcal{A}$  possiede per vincere queste sfide nei confronti di un avversario  $\mathcal{A}^{\mathcal{R}}$  che tenta di vincere le stesse sfide rispondendo in modo casuale. Ipotizziamo che  $\mathcal{A}$  abbia accesso completo a *pubLog* e che sia  $\mathcal{A}$  che  $\mathcal{A}^{\mathcal{R}}$  abbiano accesso ad una base di conoscenza comune  $\mathcal{K}_{\mathcal{A}}$  che contengono informazioni verificate su un sottoinsieme di indirizzi e i relativi proprietari.

---

## Address Unlinkability

Il seguente algoritmo descrive il meccanismo con cui avviene la sfida per la address unlinkability. In questo algoritmo si utilizza  $\mathcal{A}$  come un generico avversario, in quanto per i calcoli di quantificazione l'algoritmo verrà eseguito sia da  $\mathcal{A}$  che da  $\mathcal{A}^{\mathcal{R}}$ . Si comincia con  $\mathcal{A}$  che seleziona in modo causale un indirizzo presente in pubLog e lo invia ad uno sfidante  $\mathcal{C}$  (che si assume sia a conoscenza delle corrette correlazioni utenti-indirizzi). Se l'indirizzo scelto da  $\mathcal{A}$  è l'unico che appartiene all'utente,  $\mathcal{A}$  vince la sfida. Altrimenti  $\mathcal{C}$  sceglie un bit casuale  $b$ . Se  $b = 1$  allora  $\mathcal{C}$  sceglie un nuovo indirizzo tra quelli disponibili in pubLog che appartenga allo stesso utente del primo indirizzo, altrimenti il nuovo indirizzo verrà scelto tra quelli in pubLog che non appartengono al proprietario del nuovo indirizzo. L'indirizzo scelto insieme al precedente indirizzo vengono inviati ad  $\mathcal{A}$ , il quale stima (con un algoritmo di sua scelta, ininfluenza per la sfida) se i due indirizzi che ha ricevuto appartengono o meno allo stesso utente, e memorizza il suo risultato nel bit  $b'$ . Se  $\mathcal{A}$  ha indovinato, ovvero se  $b = b'$ , allora  $\mathcal{A}$  vince la sfida. L'algoritmo è visibile in 1.

---

### Algorithm 1 AddUnl

---

```
 $a_0 \leftarrow \mathcal{A}.\text{randomAddr}(\text{pubLog})$ 
 $\mathcal{A}.\text{send}(a_0, \mathcal{C})$ 
if  $\mathcal{C}.\text{isUniqueAddr}(a_0)$  then
     $\mathcal{A}.\text{win}()$ 
else
     $b \leftarrow \mathcal{C}.\text{randomBit}()$ 
    if  $b = 1$  then
         $a_1 \leftarrow \mathcal{C}.\text{sameUsrAddr}(a_0, \text{pubLog})$ 
    else
         $a_1 \leftarrow \mathcal{C}.\text{otherUsrAddr}(a_0, \text{pubLog})$ 
    end if
     $\mathcal{C}.\text{send}(\langle a_0, a_1 \rangle, \mathcal{A})$ 
     $b' \leftarrow \mathcal{A}.\text{estimateSameUser}(a_0, a_1)$ 
    if  $b = b'$  then
         $\mathcal{A}.\text{win}()$ 
    end if
end if
```

---

Stabiliamo che Bitcoin soddisfa il principio di *Address Unlinkability* se, per ogni avversario  $\mathcal{A}$  (che ha un ben preciso algoritmo per rispondere alla domanda di  $\mathcal{C}$ ) e per ogni coppia di indirizzi scelti dall'algoritmo,  $\mathcal{A}$  ha solo un piccolo vantaggio rispetto ad un avversario come  $\mathcal{A}^{\mathcal{R}}$  che risponde a caso alla domanda posta da  $\mathcal{C}$ . Formalmente diciamo che Bitcoin soddisfa la Address Unlinkability se:

$$\Pr[b' \leftarrow \mathcal{A}(\text{pubLog}, \mathcal{K}_{\mathcal{A}}) : b = b'] - \Pr[b' \leftarrow \mathcal{A}^{\mathcal{R}}(\mathcal{K}_{\mathcal{A}}) : b = b'] \leq \varepsilon$$

con  $\varepsilon$  trascurabile.

---

**Quantificazione di Address Unlinkability** Sfruttando i risultati ottenuti dalla sfida, è possibile stimare il *grado* con cui gli indirizzi Bitcoin possono essere collegati ad uno stesso utente. Verranno definite alcune strutture matematiche necessarie per i calcoli.

$$E_{\text{link}}[i, j] = \{p_{i,j}\}_{i,j \in [1, n_A]}$$

Rappresenta una matrice  $n_A \times n_A$  in cui ogni argomento esprime la probabilità  $p_{i,j}$  stimata da  $\mathcal{A}$  che l'indirizzo  $a_i$  appartenga allo stesso utente dell'indirizzo  $a_j$  (in simboli,  $a_i \equiv_U a_j$ ). Si definisce poi una matrice  $n_A \times n_A$  che mantenga le informazioni sulle reali connessioni tra gli indirizzi:

$$\text{GT}_{\text{link}}[i, j] = \begin{cases} 1 & \text{se } a_i \equiv_U a_j \\ 0 & \text{altrimenti} \end{cases}$$

Date queste due strutture, si definisce l'errore commesso da  $\mathcal{A}$  nella sua stima come la *distanza di Manhattan* che intercorre tra  $E_{\text{link}}[i, *]$  e le vere associazioni di  $a_i$  con tutti gli indirizzi in pubLog:

$$\begin{aligned} \text{Er}_{\mathcal{A}} &= \|E_{\text{link}}[i, *] - \text{GT}_{\text{link}}[i, *]\|_1 \\ &= \sum_x |E_{\text{link}}[i, x] - \text{GT}_{\text{link}}[i, x]| \quad \text{con } x \in [1, n_A] \end{aligned}$$

Si può dunque determinare il successo di  $\mathcal{A}$  per AddUnl come il massimo del suo errore:

$$\begin{aligned} \text{Succ}_{\mathcal{A}} &= \max_{\forall a_i \notin \mathcal{K}_{\mathcal{A}}} (\|E_{\text{link}}[i, *] - \text{GT}_{\text{link}}[i, *]\|_1) \\ &= \max_{\forall a_i \notin \mathcal{K}_{\mathcal{A}}} \left( \sum_x |E_{\text{link}}[i, x] - \text{GT}_{\text{link}}[i, x]| \right) \quad \text{con } x \in [1, n_A] \end{aligned}$$

Stesse condizioni devono essere fatte per l'avversario con criterio di decisione casuale,  $\mathcal{A}^{\mathcal{R}}$ . Mantenendo uguale  $\text{GT}_{\text{link}}[i, j]$  è necessario definire la matrice  $E^{\mathcal{R}}_{\text{link}}$  come segue:

$$E^{\mathcal{R}}_{\text{link}}[i, j] = \begin{cases} \pi_{i,j} & \text{se } \langle a_i, a_j \rangle \in \mathcal{K}_{\mathcal{A}} \\ \rho + \rho(1 - \rho)\frac{1}{2} & \text{altrimenti} \end{cases}$$

Dove  $\pi_{i,j}$  rappresenta la probabilità che gli indirizzi  $a_i$  e  $a_j$  appartengano allo stesso utente secondo  $\mathcal{K}_{\mathcal{A}}$ , mentre  $\rho$  è la frazione di indirizzi in  $\{\text{pubLog} \setminus \mathcal{K}_{\mathcal{A}}\}$  che non può essere associata ad altri indirizzi (il che capita quando un utente ha solo un indirizzo)<sup>1</sup>. Per le coppie di indirizzi non incluse in  $\mathcal{K}_{\mathcal{A}}$  tale probabilità è  $\rho + \rho(1 - \rho)\frac{1}{2}$ .

---

<sup>1</sup>l'esistenza degli indirizzi ombra non ha al momento rilevanza, ma più avanti  $\rho$  diventerà trascurabile a causa della loro presenza.



---

Le altre formule vengono mantenute uguali sostituendo  $E_{\text{link}}^{\mathcal{R}}[i, j]$  a  $E_{\text{link}}[i, j]$ . Il grado di address unlinkability risulta quindi essere il grado di successo aggiuntivo che  $\mathcal{A}$  può ottenere da pubLog in comparazione con  $\mathcal{A}^{\mathcal{R}}$ , chiamato  $Link_{\mathcal{A}}^{\text{abs}}$

$$\begin{aligned} Link_{\mathcal{A}}^{\text{abs}} &= \text{Succ}_{\mathcal{A}} - \text{Succ}_{\mathcal{A}^{\mathcal{R}}} \\ &= \frac{\text{Succ}_{\mathcal{A}} - \text{Succ}_{\mathcal{A}^{\mathcal{R}}}}{\text{Succ}_{\mathcal{A}^{\mathcal{R}}}} \quad \text{in versione normalizzata} \end{aligned}$$

### User Profile Indistinguishability

L'impossibilità di ricostruire i profili di tutti gli utenti è una proprietà molto più forte rispetto all'impossibilità di associare due indirizzi diversi ad uno stesso utente.

L'algoritmo di sfida *ProfInd* viene costruito nel modo seguente. Lo sfidante  $\mathcal{C}$  invia ad  $\mathcal{A}$  il numero  $n_U$  di utenti in  $\{\text{pubLog} \setminus \mathcal{K}_{\mathcal{A}}\}$ , che risponde con  $n_U$  insiemi di indirizzi (o transazioni) e con la matrice  $E_{\text{prof}} = \{g_i\}_{i=1}^{n_U}$  rappresentante la stima fatta da  $\mathcal{A}$  sui profili degli utenti nel sistema. Come per l'algoritmo precedente, definiamo  $\text{GT}_{\text{prof}} = \{\text{gt}_i\}_{i=1}^{n_U}$  che rappresenta le vere associazioni tra indirizzi (o transazioni) e utenti, per cui:

$$\text{GT}_{\text{prof}} = \begin{cases} \{a_{u_i}\}_{i=1}^{n_U} & \text{per profili basati su indirizzi} \\ \{\tau_{u_i}\}_{i=1}^{n_U} & \text{per profili basati su transazioni} \end{cases}$$

dove  $a_{u_i}$  e  $\tau_{u_i}$  rappresentano insiemi di indirizzi/transazioni per l'utente  $u_i$ . Ovviamente,  $\mathcal{A}$  vince la sfida se indovina correttamente il profilo, ovvero  $E_{\text{prof}} \equiv \text{GT}_{\text{prof}}$ .

---

#### Algorithm 2 ProfInd

---

```

 $\mathcal{C}.\text{send}(n_U, \mathcal{A})$ 
 $E_{\text{prof}} \leftarrow \mathcal{A}.\text{estimateProfile}()$ 
if  $E_{\text{prof}} = \text{GT}_{\text{prof}}$  then
   $\mathcal{A}.\text{win}()$ 
end if

```

---

Diciamo che un sistema soddisfa la proprietà di *Profile Indistinguishability* se non esiste nessun avversario  $\mathcal{A}$  in grado di vincere *ProfInd* con una probabilità migliore dell'avversario che risponde casualmente  $\mathcal{A}^{\mathcal{R}}$ , ovvero:

$$\begin{aligned} \forall \mathcal{A} : \Pr[E_{\text{prof}} \leftarrow \mathcal{A}(\text{pubLog}, n_U) : E_{\text{prof}} \equiv \text{GT}_{\text{prof}}] - \\ \Pr[E_{\text{prof}}^{\mathcal{R}} \leftarrow \mathcal{A}^{\mathcal{R}}(n_U) : E_{\text{prof}}^{\mathcal{R}} \equiv \text{GT}_{\text{prof}}] \leq \varepsilon \end{aligned}$$

**Quantificazione di Profile Indistinguishability** Anche in questa stima come per la precedente, la quantificazione dipende dal confronto tra la stima di  $\mathcal{A}$  e i dati reali. Definiamo quindi la similitudine tra  $E_{\text{prof}}$  e  $\text{GT}_{\text{prof}}$  come la funzione  $\text{Sim}(E_{\text{prof}}, \text{GT}_{\text{prof}})$ , i cui valori appartengono all'insieme  $[0, 1]$ . Come per la address

---

unlinkability, misurano la profile indistinguishability contro  $\mathcal{A}$  stimando il grado con cui  $\mathcal{A}$  è in grado di stilare un profilo utente corretto, ovvero misurando il vantaggio che  $\mathcal{A}$  ha rispetto a  $\mathcal{A}^{\mathcal{R}}$  nell'avvicinarsi a  $\text{GT}_{\text{prof}}$ :

$$\text{Prof}_{\mathcal{A}} = \text{Sim}(E_{\text{prof}}, \text{GT}_{\text{prof}}) - \text{Sim}(E_{\text{prof}}^{\mathcal{R}}, \text{GT}_{\text{prof}})$$

Per quantificare  $\text{Sim}(E_{\text{prof}}, \text{GT}_{\text{prof}})$  e  $\text{Prof}_{\mathcal{A}}$  è necessario sfruttare alcune metriche per calcolare le distanze basate sull'entropia, la Normalized Mutual Information (NMI) e la Adjusted Mutual Information (AMI). La NMI valuta la similarità di due raggruppamenti degli stessi oggetti e assume valori tanto più alti (il massimo è 1) tanto più i due raggruppamenti sono identici. La AMI, dati due raggruppamenti  $G_1$  e  $G_2$ , si avvicina allo 0 quando  $G_1$  risulta simile ad un raggruppamento casuale, mentre si avvicina a 1 quando  $G_1 = G_2$ . Quindi AMI valuta direttamente il vantaggio che ha  $\mathcal{A}$  nel vincere la sfida di *ProfInd*.

Nel caso di profili basati sugli indirizzi, NMI e AMI sono calcolate come segue:

$$\begin{aligned} \text{NMI} &= \frac{\mathcal{I}(E_{\text{prof}}, \text{GT}_{\text{prof}})}{\max(H(E_{\text{prof}}), H(\text{GT}_{\text{prof}}))} \\ \text{AMI} &= \frac{\mathcal{I}(E_{\text{prof}}, \text{GT}_{\text{prof}} - \mathcal{E})}{\max(H(E_{\text{prof}}), H(\text{GT}_{\text{prof}})) - \mathcal{E}} \end{aligned}$$

dove:

$$\begin{aligned} \mathcal{I}(E_{\text{prof}}, \text{GT}_{\text{prof}}) &= \sum_{i=1}^{n_U} \left( \sum_{j=1}^{n_U} \left( \frac{n_{(i,j)}}{n_A} \log \left( \frac{n_{(i,j)} \cdot n_A}{n_{(i,*)} n_{(*,j)}} \right) \right) \right) \\ H(E_{\text{prof}}) &= - \sum_{i=1}^{n_U} \left( \frac{n_{(i,*)}}{n_A} \log \left( \frac{n_{(i,*)}}{n_A} \right) \right) \\ H(\text{GT}_{\text{prof}}) &= - \sum_{j=1}^{n_U} \left( \frac{n_{(*,j)}}{n_A} \log \left( \frac{n_{(*,j)}}{n_A} \right) \right) \\ \mathcal{E} &= \sum_{i=1}^{n_U} \left( \sum_{j=1}^{n_U} \left( \sum_{n \in \mathcal{M}} \left( \frac{n}{n_A} \log \left( \frac{n_A n}{n_{(i,*)} n_{(*,j)}} \right) \frac{n_{(i,*)}! n_{(*,j)}! (n_A - n_{(i,*)})! (n_A - n_{(*,j)})!}{n_A! (n_{(i,*)} - n)! (n_{(*,j)} - n)! (n_A - n_{(i,*)} - n_{(*,j)} - n)!} \right) \right) \right) \\ \mathcal{M} &= [\max(n_{(i,*)} + n_{(*,j)} - n_A, 0), \min(n_{(i,*)}, n_{(*,j)})] \end{aligned}$$

In queste formule,  $n_A$  è il numero di indirizzi Bitcoin,  $n_{(i,j)}$  è il numero di indirizzi di  $u_i$ , che vengono assegnati al gruppo  $g_j$ ,  $n_{(i,*)}$  e  $n_{(*,j)}$  sono il numero di indirizzi di  $u_i$  e  $g_i$  rispettivamente.  $\mathcal{E}$  rappresenta l'informazione mutuale attesa tra il raggruppamento  $\text{GT}_{\text{prof}}$  e il raggruppamento casuale di indirizzi  $E_{\text{prof}}^{\mathcal{R}}$ . Si possono ottenere risultati simili per calcolare NMI e AMI nel caso di profili basati sulle transazioni.

È importante far presente che, sebbene NMI e AMI siano efficaci per rappresentare il successo di  $\mathcal{A}$  nella creazione di un profilo per tutti gli utenti della rete, non sono in grado di misurare il successo dell'avversario nel creare un profilo per un utente specifico. Nella sezione 3.2.4 verrà misurato il successo di  $\mathcal{A}$  nel creare il profilo di un utente specifico  $u_i$  stabilendo la massima similitudine degli indirizzi (o transazioni) di  $u_i$  con ogni cluster avversario, ovvero  $\max_{\forall j} (\text{Sim}(a_{u_i}, g_j))$ .

---

## 3.2 Applicare il modello

Vediamo ora come il nostro avversario, dato pubLog, può raccogliere informazioni sugli utenti Bitcoin sfruttando alcune proprietà delle attuali implementazioni del client Bitcoin. Grazie ad un simulatore vedremo poi quanto il modello precedentemente descritto risulti valido data la conoscenza acquisita.

### 3.2.1 Euristiche per sfruttare il client

**Transazioni multi-input** Quando per una transazione non è possibile sfruttare l'output di una singola transazione precedente, è necessario usare come input più di una transazione. I client scelgono un insieme di BTC dal portafoglio dell'utente in modo che il loro valore totale sia quello richiesto dalla transazione ed effettuano così una transazione multi-input. Dato che le BTC sono nel portafoglio dell'utente, se tali BTC sono prese da indirizzi diversi, allora tali indirizzi appartengono tutti allo stesso utente.

**Indirizzi ombra** Come detto in precedenza, gli indirizzi ombra vengono creati dal client per raccogliere l'eventuale resto delle transazioni. Immaginando quindi una transazione con un input e due output, si può ragionevolmente stabilire quale dei due indirizzi sia quello ombra appartenente allo stesso utente che ha inviato la transazione. L'indirizzo ombra sarà infatti creato sul momento, dunque sarà l'indirizzo che non è presente nel pubLog in data precedente alla transazione. Questo è valido se si assume che gli utenti della rete siano attivi (abbiano fatto almeno una transazione) e che il client non consenta di creare una transazione con più indirizzi in output.

Per valutare la bontà delle euristiche, creiamo un parser che analizzi alcuni blocchi e raggruppi gli indirizzi in cluster di indirizzi generici (GA) basati sulle euristiche.

Lanciando il parser sui primi 140000 blocchi della blockchain (ovvero i blocchi creati fino ad Agosto 2011), il parser restituisce 1632648 indirizzi unici. Utilizzando la prima euristica, tali indirizzi possono essere classificati in 1069699 distinti GA, mentre con la seconda euristica i numeri di distinti GA scende a 693051. Dato che a Settembre 2011 esistevano circa 60000 utenti Bitcoin, i risultati ottenuti indicano che è stato raggruppato circa il 58% degli indirizzi Bitcoin con una media di 11.55 indirizzi per cluster. Ciò dimostra che raccogliendo informazioni con queste euristiche, il vantaggio di  $\mathcal{A}$  per AddUnl è considerevole.

### 3.2.2 Analisi basata sul comportamento

Ora come ora, esistono molteplici client per la rete Bitcoin, e anche quello ufficiale va spesso incontro a modifiche e miglioramenti. Quello che difficilmente cambia è il comportamento degli utenti. Esistono alcune tecniche di raggruppamento basate sul comportamento che  $\mathcal{A}$  può sfruttare, come ad esempio gli algoritmi K-Means Clustering (KMC) e Hierarchical Agglomerative Clustering (HAC). Definiamo  $U$  come l'insieme di tutti gli utenti Bitcoin, e  $(GA_1, \dots, GA_{n_{GA}})$  i raggruppamenti ottenuti da

---

$\mathcal{A}$  applicando le due euristiche precedentemente descritte a pubLog. Con questi dati, l'obiettivo di  $\mathcal{A}$  è ottenere un gruppo di clusters di indirizzi  $E_{\text{prof}} = \{g_1, \dots, g_{n_U}\}$  che approssimi il più possibile  $U$ . Dato che ogni GA è posseduta da esattamente un utente, la stima dell'assegnamento di ogni GA può essere modellata con una variabile  $z_i$  tale che  $z_i = k \iff \text{GA}_i$  appartiene a  $g_k$ . L'algoritmo HAC assume che inizialmente ogni GA rappresenti un utente separato ( $\{z_i = i\}_{i=1}^{n_{\text{GA}}}$ ) e calcola valori di similitudine per ogni coppia di cluster. I cluster con elevata similarità vengono raggruppati insieme e il processo si ripete sui nuovi raggruppamenti, fermandosi quando il numero di raggruppamenti è esattamente uguale al numero di utenti  $n_U$ . A questo punto si avvia KMC, inizializzato con l'output di HAC, il quale assume che ogni utente sia rappresentato dal punto centrale di ogni raggruppamento. L'algoritmo tenta di minimizzare la distanza tra i GA e i cluster a cui essi sono stati assegnati, ricalcolando ad ogni round sia il centro dei cluster sia la distanza GA-cluster.

Nella loro implementazione, gli autori di [?] hanno rappresentato ogni transazione all'interno delle GA con tre punti:

1. L'orario in cui la transazione ha avuto luogo.
2. Gli indici dei diversi GA che appaiono all'interno della transazione come mittenti o destinatari.
3. La quantità di BTC spese nella transazione.

$\tau_x$  identifica l'insieme delle transazioni di  $\text{GA}_x$ , e il grado di similitudine tra due cluster  $\text{GA}_i$  e  $\text{GA}_j$  è rappresentato dal coseno di similitudine <sup>2</sup> tra le liste  $\tau_i$  e  $\tau_j$ , ovvero:

$$\text{Sim}^{\text{hac}}(\text{GA}_i, \text{GA}_j) = \frac{\sum_{\forall \tau \in \tau_i \cap \tau_j} (f_{(\tau,i)} \cdot f_{(\tau,j)})}{\|\tau_i\|_2 \cdot \|\tau_j\|_2}$$

dove  $f_{(\tau,x)}$  sono le occorrenze dell'oggetto  $\tau$  nel vettore  $\tau_x$ . Da ciò si deriva la metrica per la distanza in KMC:

$$\text{Dist}^{\text{kmc}}(\text{GA}_i, g_k) = \frac{2}{1 + \text{textSim}^{\text{hac}}(\text{GA}_i, g_k)} - 1$$

L'implementazione tiene anche conto di alcune restrizioni imposte dal mondo reale. In particolare, dato che gli utenti non possono fisicamente essere in due posti allo stesso tempo, non possono partecipare in due distinti (fisicamente) scambi di beni allo stesso tempo.

### 3.2.3 Simulazione: utilizzo di Bitcoin in ambiente universitario

Per verificare i loro modelli, i ricercatori hanno simulato l'utilizzo di Bitcoin per le transazioni quotidiane da parte degli utenti del Dipartimento di Informatica all'ETH di Zurigo con l'assunzione che i negozi nei dintorni della facoltà accettino BTC come valuta.

---

<sup>2</sup>Tecnica euristica che misura la differenza tra due vettori, spesso usata per il confronto dei testi

---

**Configurazione della simulazione** Il simulatore riceve in input un file XML come input e output:

- un log che descrive gli eventi simulati, ovvero la *base di verità*.
- il pubLog risultante dalla simulazione.

I file XML di configurazione contengono tutti i parametri necessari alla simulazione, come il numero di utenti, il numero di nodi che generano i blocchi (i *miners*), il tempo di simulazione, la difficoltà della generazione dei blocchi, le configurazioni per creare i profili utente e i venditori/acquirenti. Gli output del simulatore vengono utilizzati per valutare il successo di  $\mathcal{A}$ : un parser Perl utilizza i blocchi Bitcoin simulati come input ed effettua una prima classificazione degli indirizzi simulati in GA in base alle due euristiche descritte. Questo output prefiltrato viene dato in pasto agli algoritmi KMC e HAC (entrambi implementati in C). L'output di tali algoritmi viene infine comparato con la base di verità tramite un secondo script Perl che calcola il successo di  $\mathcal{A}$  sulle sfide AddUnl e ProfInd.

La configurazione utilizzata rappresenta uno scenario realistico per gli studenti e lo staff del Dipartimento di Informatica dell'Università di Zurigo durante il semestre autunnale del 2012. Si è simulato un numero variabile di utenti dei quali il 5.2% sono *Professori*, il 42.0% fanno parte dello *Staff* e il restante 52.8% sono *Studenti*. Vengono considerati in tutto 6 eventi distinti con diverse varianti:

- Pranzo e cena (12 varianti).
- Acquisto di alimentari (2 varianti).
- Acquisto da distributori automatici (4 varianti).
- Shopping online (5 varianti).
- Acquisto di libri (2 varianti).
- Baratto con altri utenti.

Ad ogni utente viene assegnata una probabilità per ognuna delle varianti in accordo alla categoria (Professore, Staff, Studente) a cui appartiene. Per ogni evento vengono stabilite la frequenza con cui avviene e un range di prezzo, mentre alle varianti viene assegnato un punteggio in base alla loro popolarità. La probabilità che una variante venga selezionata è il risultato dell'interpolazione tra la frequenza degli eventi in una settimana e il punteggio della variante. Per assicurare una grande varietà di profili utente, nella configurazione per ogni evento vengono specificati valori massimi e minimi per la frequenza, il punteggio e il prezzo. Tali limiti dipendono dalla categoria dell'utente, dall'evento e dalla variante in questione. All'inizio della sperimentazione, gli utenti hanno meno di 10 indirizzi Bitcoin a testa. Man mano che vengono effettuate transazioni, nuovi indirizzi ombra vengono creati nei loro portafogli. Nella configurazione è stato inoltre modellato il comportamento degli

---

utenti attenti alla propria privacy: tali utenti periodicamente creano nuovi indirizzi nei propri portafogli e trasferiscono parte delle proprie BTC dal vecchio al nuovo indirizzo.

### 3.2.4 Risultati della simulazione

Ad ogni round della simulazione sono stati emulati due diversi scenari. Nel primo scenario, detto di “*Conoscenza Parziale*” è stato assunto che  $\mathcal{A}$  sia consapevole della posizione e del servizio offerto da tutti i commercianti che accettano Bitcoin e possa quindi comprendere quando una transazione è il risultato di un materiale scambio di beni. In tal caso, l’indirizzo del commerciante è stato inserito in  $\mathcal{K}_{\mathcal{A}}$  durante la computazione di AddUnl. Si è inoltre assunto che  $\mathcal{A}$  sia in grado di ottimizzare l’algoritmo di clustering in modo da tenere conto del fatto che un utente che effettua una transazione da qualche parte non può effettuare un’altra transazione altrove nello stesso momento (ovviamente ciò è valido nel caso in cui  $\mathcal{A}$  sia in grado di stabilire la posizione fisica degli utenti coinvolti nella transazione, il che avviene se pubLog contiene tali informazioni). Il secondo scenario si contrappone al primo in quanto basato su “*Conoscenza Zero*”, ovvero  $\mathcal{A}$  non conosce la posizione di utenti e commercianti e quindi non ha nessuna informazione a priori, ma assume arbitrariamente che al massimo il 10% delle transazioni avvenga come risultato di uno scambio di beni ordinati via Internet.

Con questo setup vengono valutate le metriche per  $\text{Link}_{\mathcal{A}}$ ,  $\text{Prof}_{\mathcal{A}}^{\mathcal{I}}$  per i profili basati sugli indirizzi e  $\text{Prof}_{\mathcal{A}}^{\mathcal{T}}$  per i profili basati sulle transazioni rispetto al numero totale degli utenti  $n_U$  e la frazione di utenti attenti alla privacy che generano manualmente nuovi indirizzi e vi spostano parte delle BTC. I risultati nella tabella 3.1 mostrano come non ci siano grandi differenze tra i due diversi segnali, il che vuol dire che poco cambia per  $\mathcal{A}$  il fatto di sapere qualcosa di più o di meno sugli utenti di cui stilare il profilo.

In entrambi gli scenari, il vantaggio di  $\mathcal{A}$  nell’associare indirizzi e utente è influenzato solamente in minima parte dalla frazione di utenti che si interessano della loro privacy, surclassando  $\mathcal{A}^{\mathcal{R}}$  di almeno il 90%. Per quanto riguarda la creazione di profili utente invece, gli utenti che creano indirizzi manualmente sono più influenti: se non sono presenti tali utenti, il vantaggio per la creazione di profili basati sugli indirizzi si attesta intorno all’88%, mentre per i profili basati sulle transazioni è intorno al 73%. Aumentando però la percentuale di utenti sensibili alla privacy, il vantaggio diminuisce al 70% per i profili sugli indirizzi e al 63% per i profili sulle transazioni. La spiegazione è che la creazione continua di nuovi indirizzi genera rumore nel log di Bitcoin, complicando il lavoro degli algoritmi. In ogni caso, dato che i risultati di AMI sono sempre più vicini al 100% che non allo 0% indica come  $\mathcal{A}$  in generale abbia performance molto migliori di  $\mathcal{A}^{\mathcal{R}}$  rappresentando raggruppamenti più simili alla realtà.

Tabella 3.1: Risultati del clustering basato sul comportamento. Le colonne titolate **X (Y%)** indicano una simulazione effettuata con **X** utenti di cui il **Y%** sono attenti alla propria privacy. Ogni valore è la media di 5 diverse iterazioni ed è rappresentato con intervalli di confidenza del 95% dopo il segno  $\pm$ .

Utenti:		100 (50%)	200 (0%)	200 (50%)	200 (100%)	400 (59%)
<b>Link<sub>A</sub></b>		0.91 $\pm$ 0.01	0.90 $\pm$ 0.01	0.91 $\pm$ 0.01	0.92 $\pm$ 0.01	0.93 $\pm$ 0.01
<b>Prof<sub>A</sub><sup>a</sup></b>	NMI	0.76 $\pm$ 0.01	0.87 $\pm$ 0.01	0.79 $\pm$ 0.01	0.70 $\pm$ 0.01	0.80 $\pm$ 0.01
	AMI	0.75 $\pm$ 0.01	0.86 $\pm$ 0.01	0.77 $\pm$ 0.01	0.68 $\pm$ 0.01	0.77 $\pm$ 0.01
<b>Prof<sub>A</sub><sup>r</sup></b>	NMI	0.68 $\pm$ 0.01	0.79 $\pm$ 0.02	0.70 $\pm$ 0.01	0.65 $\pm$ 0.01	0.72 $\pm$ 0.01
	AMI	0.67 $\pm$ 0.01	0.72 $\pm$ 0.01	0.69 $\pm$ 0.01	0.63 $\pm$ 0.01	0.70 $\pm$ 0.01

(a) Risultati dello scenario a **conoscenza parziale**

Utenti:		100 (50%)	200 (0%)	200 (50%)	200 (100%)	400 (59%)
<b>Link<sub>A</sub></b>		0.90 $\pm$ 0.01	0.90 $\pm$ 0.01	0.91 $\pm$ 0.01	0.92 $\pm$ 0.01	0.93 $\pm$ 0.01
<b>Prof<sub>A</sub><sup>a</sup></b>	NMI	0.79 $\pm$ 0.01	0.89 $\pm$ 0.01	0.79 $\pm$ 0.01	0.71 $\pm$ 0.02	0.80 $\pm$ 0.01
	AMI	0.78 $\pm$ 0.02	0.88 $\pm$ 0.01	0.78 $\pm$ 0.02	0.69 $\pm$ 0.02	0.78 $\pm$ 0.01
<b>Prof<sub>A</sub><sup>r</sup></b>	NMI	0.69 $\pm$ 0.01	0.73 $\pm$ 0.03	0.69 $\pm$ 0.03	0.65 $\pm$ 0.01	0.72 $\pm$ 0.01
	AMI	0.68 $\pm$ 0.01	0.72 $\pm$ 0.01	0.68 $\pm$ 0.03	0.63 $\pm$ 0.01	0.70 $\pm$ 0.01

(b) Risultati dello scenario a **conoscenza zero**

Il grafico 3.1 descrive la porzione di utenti<sup>3</sup> che vengono “catturati” da  $\mathcal{A}$ . Si nota come, nel caso di 200 utenti nessuno dei quali interessato alla propria privacy, quasi il 42% viene inserito in un cluster con un’accuratezza dell’80%. Nel caso tutti gli utenti siano sensibili alla propria privacy, solo il 35% di essi viene raggruppato con un’accuratezza di almeno l’80%. Si può dunque concludere che la privacy di un utente può essere compromessa anche nel caso in cui esso crei nuovi indirizzi per gestire le proprie BTC.

Figura 3.1: Percentuale di transazioni catturate dagli algoritmi di cluster nello scenario della **conoscenza parziale**.

I risultati mostrano inoltre come il vantaggio di  $\mathcal{A}$  rispetto ad  $\mathcal{A}^{\mathcal{R}}$  per tentare di collegare gli indirizzi di uno stesso utente, non è influenzata in modo rilevante dal numero di utenti della rete. Nel caso della creazione dei profili invece aumentando il numero di utenti da 100 a 400, il successo aumenta da 76% per gli indirizzi e 68% per transazioni a 80% e 72% rispettivamente. Ciò è dovuto principalmente al fatto che con l’aumentare del numero di utenti, l’assegnamento di indirizzi (o transazioni) a gruppi di utenti diventa molto meno efficace per  $\mathcal{A}^{\mathcal{R}}$ .

<sup>3</sup>misurata secondo i criteri di similarità delle transazioni che appaiono in un portafogli utente e i corrispondenti cluster che sono stati discussi nel paragrafo 3.1.2

---

Nel grafico 3.2 viene valutato il caso in cui  $\mathcal{A}$  non è riuscito ad ottenere una stima accurata del numero di utenti totali presenti nello scenario. Ciò non influenza l'efficacia del clustering comportamentale, permettendo lo stesso di compromettere una significativa porzione di utenti.

Figura 3.2: Caso in cui  $\mathcal{A}$  non può stimare accuratamente  $n_U = 200$  nello scenario di **conoscenza parziale**.

Infine il grafico 3.3 mostra come la percentuale di transazioni correttamente catturate dagli algoritmi di raggruppamento non sono influenzate dal numero di utenti del sistema: ad esempio la percentuale di utenti “catturati” con una accuratezza dell'80% e circa il 40% quando  $n_U = 100, 200, 400$ .

Figura 3.3: Porzione delle transazioni catturate da  $\mathcal{A}$  nello scenario di **conoscenza zero** e con il 50% degli utenti attenti alla propria privacy.

### 3.3 Caso reale: analisi di un furto

Tutti questi risultati sono stati ottenuti in un contesto universitario ben delimitato come estensione possibili variabili, per cui essi rappresentano più il limite superiore di una corretta valutazione della privacy del sistema Bitcoin piuttosto che una precisa descrizione di essa nella rete reale. Le tecniche illustrate sono state però applicate, dopo essere state adattate, alla rete reale dai ricercatori Fergal Reid e Martin Harrigan in [?], i quali hanno anche utilizzato gli strumenti da loro creati per analizzare un reale caso di furto di Bitcoin.

#### 3.3.1 Mappare la rete Bitcoin

Come visto, la disponibilità al pubblico dello storico delle transazioni, le relazioni tra esse e il modo in cui si gestiscono i propri indirizzi<sup>4</sup> permette di analizzare la rete Bitcoin e di raggruppare le informazioni ivi contenute. In particolare Redi ed Harrigan sfruttano due distinti grafi realizzabili a partire dalle informazioni pubblicamente disponibili, un grafo che rappresenta le connessioni tra le transazioni e uno che rappresenta le connessioni tra gli utenti. Il primo grafo viene definito *transaction network* ed illustra il flusso di BTC nel tempo: ogni vertice è una transazione ed ogni arco orientato rappresenta l'output della transazione sorgente e l'input della transazione destinataria, il tutto etichettato con il valore della transazione e un timestamp. Il secondo grafo, lo *user network*, illustra il flusso di BTC tra gli utenti: ogni vertice è un utente e ogni arco orientato rappresenta una transazione in cui

---

<sup>4</sup>Si ricorda che ogni indirizzo è ricavato da una chiave pubblica, la quale è collegata ad una chiave privata custodita (si spera gelosamente) da un utente.



Tabella 3.2: Distribuzioni del grado dei nodi per la transaction network

Variabile	$\tilde{x}$	$\bar{x}$	s	$\alpha$	$x_{\min}$	GoF	p-valore
Grado	3	3.20	6.20	3.24	50	0.02	0.05
Grado in ingresso	1	1.60	5.31	2.50	4	0.01	0.00
Grado in uscita	1	1.60	3.17	3.50	51	0.05	0.00

l'indirizzo sorgente è associato all'utente/vertice di partenza e l'indirizzo destinatario è associato all'utente/vertice di arrivo. Per associazione si intende che la chiave privata associata all'indirizzo/chiave pubblica è in possesso dell'utente o è comunque a lui accessibile. Per realizzare tali reti sono stati impiegati i dati fino all'ultima transazione avvenuta il 12 Luglio 2011, raccolti con una versione modificata dello strumento `bitcointools`<sup>5</sup> di Gavin Andresen. Il dataset comprende quindi 1019486 transazioni tra 1253054 distinti indirizzi.

### Transaction Network

A partire dal dataset, la costruzione della transaction network non richiede nessuna elaborazione dei dati. La rete ha 975520 vertici e 1558854, in quanto sono state omesse transazioni non connesse ad almeno un'altra transazione<sup>6</sup>. Data la natura delle transazioni, la rete è priva di multi-archi (archi tra gli stessi nodi con lo stesso verso) e di cicli, risultando quindi in un Grafo Direzionale Aciclico (DAG).

Si è studiata la distribuzione cumulativa<sup>7</sup> del grado di ogni nodo (il numero di connessioni entranti o uscenti dal nodo) secondo la legge di potenza  $p(x) \propto x^{-\alpha}$  per  $x > x_{\min}$ . I parametri di tale funzione sono stati stimati sfruttando una metodologia Goodness-of-Fit le cui statistiche e p-valori sono visibili nella tabella ?? insieme alle stime dei parametri per le distribuzioni del grado (complessivo, in ingresso e in uscita) dei nodi. Si osserva che nessuna delle distribuzioni studiata ha la legge di potenza come possibile ipotesi ( $p > 0.1$ ). Ciò è dovuto al fatto che non esiste nessun collegamento preferenziale: i nuovi vertici si uniscono ai vertici esistenti le cui transazioni non sono ancora state del tutto reclamate, ovvero ai nodi che hanno ancora BTC da spendere. Il grafo presenta 1949 componenti debolmente connesse, in cui il componente più grande possiede 948287 vertici (il 97.31%) che possiede anche un componente biconnesso con 716354 vertici, il 75.64% di tutti i vertici del componente che lo contiene. Se si analizza anche il numero di nodi, la densità e la lunghezza media dei cammini, la crescita e la dispersione della rete nel tempo risultano evidenti.

<sup>5</sup><https://github.com/gavinandresen/bitcointools>

<sup>6</sup>Ovvero le transazioni risultanti dal mining e le transaction fee.

<sup>7</sup>frazione dei dati con valore in esame  $\geq k$ , in questo caso frazione dei nodi il cui grado è  $\geq k$ .

Tabella 3.3: Distribuzioni del grado dei nodi per la user network

Variabile	$\tilde{x}$	$\bar{x}$	s	$\alpha$	$x_{\min}$	GoF	p-valore
Grado	3	4.45	218.10	2.38	66	0.02	0.00
Grado in ingresso	1	2.22	86.40	2.45	57	0.05	0.00
Grado in uscita	2	2.22	183.91	2.03	10	0.22	0.00

## User Network

Tale rete rappresenta le transazioni che intercorrono tra gli utenti nel tempo, in cui i vertici sono gli utenti e gli archi diretti sono le transazioni etichettate con ammontare di BTC e timestamp. Come precedentemente visto, le informazioni sugli utenti non sono direttamente visibili dal dataset, per cui è necessario effettuare una fase di preprocessing in cui raggruppare gli indirizzi di uno stesso utente sfruttando le tecniche precedentemente illustrate. Si immagini quindi di partire da un grafo incompleto contenente soltanto gli indirizzi e non gli utenti e di cercare di contrarre tali vertici sfruttando l'euristica multi-input precedentemente illustrata. Si crei quindi una rete subordinata con indirizzi come vertici, e si connettano i vertici che risultano essere input della stessa transazione (e quindi appartenenti allo stesso utente) tramite archi non direzionati. Questa rete subordinata risulta avere 1253054 vertici e 4929950 archi e, cosa fondamentale, contiene 86641 componenti connesse. Ognuna di queste componenti corrisponde ad un utente ed ogni vertice di un componente è un indirizzo del relativo utente. Alla fine della fase di preprocessing, la user network risulta avere 881678 vertici (86641 componenti connesse e 795037 vertici isolati della rete ausiliaria) e 1961636 archi diretti. La rete è ancora incompleta, ma per il tipo di analisi che si intende descrivere è una approssimazione più che sufficiente. Ci sono 604 componenti debolmente connesse e 579355 componenti fortemente connesse<sup>8</sup>, il componente più grande conta 879859 (il 99.79% dei vertici totali) e contiene anche un componente biconnesso con 652892, il che rappresenta il 74.20% dei vertici del componente gigante. A differenza della rete delle transazioni, questa è una rete con multi-archi e cicli. Anche qui sono state fatte analisi sulla distribuzione del grado dei nodi, con risultati visibili nella tabella ??, le quali mostrano che come nella rete delle transazioni nessuna delle leggi di potenza sperimentate è una ipotesi plausibile per descrivere la distribuzione. Il tasso di distribuzione e di crescita della rete è in crescita nel tempo esattamente come la transaction network.

La rete degli utenti appena creata è sotto molti aspetti anche una mappa della rete sociale di Bitcoin. Analizzandola è quindi possibile identificare comunità e utenti centrali che esistono nella rete. Dal punto di vista dell'anonimato, ci si aspetterebbe di trovare delle reti simili ad alberi che rappresentino il flusso di BTC tra indirizzi utilizzati una sola volta e non collegabili ad altri indirizzi. I risultati mostrano invece una struttura con un numero considerevole di cicli, il che semplifica notevolmente

<sup>8</sup>Esiste un arco orientato tra ogni coppia di nodi.

---

l'analisi della rete alla ricerca di un utente specifico, soprattutto se associata a dati importati da fonti off-network.

### **Integrare user network con informazioni off-network**

Sebbene la rete non contenga di per se alcuna informazione riguardo l'utente, esistono numerosi (e molto utilizzati) servizi integrati nella rete che richiedono informazioni che possono identificare l'utente, quali indirizzi email, indirizzi di spedizione, carte di credito, indirizzi IP, ecc. Nel momento in cui qualcuna di queste informazioni diventa pubblicamente accessibile<sup>9</sup>, allora anche l'identità degli utenti coinvolti tramite transazioni con gli utenti, le cui generalità sono state scoperte, sono anch'esse a rischio<sup>10</sup>. Esistono varie situazioni in cui tali informazioni possano essere rese pubbliche.

**The Bitcoin Faucet** *The Bitcoin Faucet*<sup>11</sup> è un servizio a cui gli utenti possono donare BTC in modo che vengano distribuite in piccole quantità ad altri utenti. Per evitare abusi, il servizio pubblica uno storico delle recenti donazioni insieme agli indirizzi IP dei beneficiari. Esiste quindi la possibilità di associare un indirizzo IP ad un indirizzo Bitcoin, ed analizzando nel tempo la pagina si può realizzare una mappatura tra utenti e IP. Reid e Harrigan si sono accorti di come molti degli indirizzi bitcoin che hanno ricevuto BTC possano essere contratti con altri indirizzi bitcoin nella rete ausiliaria, rivelando quindi indirizzi IP collegati in qualche modo a precedenti transazioni al di fuori del servizio. Essendo gli indirizzi IP geolocalizzati, è stata creata una mappa degli indirizzi bitcoin che hanno ricevuto BTC nel periodo di una settimana. La figura ?? mostra tale mappa incrociata con i dati della user network: un arco tra due indirizzi IP indica che gli utenti corrispondenti sono collegato da un cammino (diretto o indiretto) di lunghezza  $\leq 3$  nella user network<sup>12</sup>. Tale figura deve servire come prova di un concetto: se i dati resi pubblici da una realtà piccola come Bitcoin Faucet possono portare ad un simile risultato, i risultati ottenibili sfruttando i dati posseduti (non necessariamente divulgati) da una grande realtà potrebbero essere decisamente critici per la privacy dell'utente.

**Rivelazioni volontarie** In molti forum relativi alla rete Bitcoin (come ad esempio il forum ufficiale <http://forum.bitcoin.org>) ma anche siti web e social network, è costume degli utenti pubblicare in varie forme il proprio indirizzo Bitcoin, spesso con lo scopo di ricevere donazioni da altri utenti. Dato che gli indirizzi sono semplici stringhe alfanumeriche di 27-34 caratteri, i motori di ricerca li indicizzano perfettamente. In molti casi, Reid ed Harrigan sono riusciti ad recuperare informazioni su alcuni indirizzi con una semplice ricerca di tali indirizzi su Google, e sfruttando

---

<sup>9</sup>oppure è accessibile dalle forze dell'ordine a seguito di apposito mandato

<sup>10</sup>a meno che non si siano usati strumenti per mascherare l'IP come proxy o la rete TOR

<sup>11</sup><http://freebitcoins.appspot.com>

<sup>12</sup>Il cammino in questione non deve includere The Bitcoin Faucet.

---

la user network secondaria hanno recuperato molti altri indirizzi associati a quelli recuperati con una semplice visita sul web.

**Analisi dello stack TCP/IP** Il ricercatore Dan Kaminsky ha individuato una falla di sicurezza nella rete a livello TCP/IP: se si modifica un client bitcoin in modo da connetterlo ad ogni altro nodo della rete, è possibile mappare gli indirizzi Bitcoin agli indirizzi IP basandosi sull'assunzione che il primo nodo ad informare gli altri di una transazione è il nodo che ha realizzato tale transazione. Con tale approccio non è necessario cercare i dati da fonti esterne, basta modificare avere un client apposito simile a quello creato da Decker e Wattenhofer (2.8.2).

### 3.3.2 Analisi di un furto

Con la rete da loro creata, Harrigan e Reid hanno analizzato un furto denunciato sul forum ufficiale di Bitcoin<sup>13</sup> da un utente chiamato *allinvain*. Sembra che in data 13/06/2011 circa 25000 BTC siano state inviate a *pk<sub>red</sub>*<sup>14</sup> alle ore 16:52:23. Il furto è avvenuto poco dopo che qualcuno si è introdotto nell'account della vittima sulla mining pool Slush<sup>15</sup> e ha modificato l'indirizzo di pagamento da *pk<sub>green</sub>*<sup>16</sup> a *pk<sub>blue</sub>*<sup>17</sup>. All'epoca del furto, le BTC rubate avevano un valore di mercato di circa mezzo milione di dollari americani, il che può far capire quanto elevato fosse all'epoca la necessità del ladro di restare anonimo.

Si prenda la user network incompleta senza nessuna contrazione dei vertici e ci si concentri sulla *rete egocentrica* che circonda il ladro, ovvero tutti i nodi a distanza massima 2 (senza tenere conto del verso degli archi) e tutti gli archi indotti da tali vertici. Per evitare confusione vengono rimossi anche tutti i cicli, i multi-archi e gli archi che non sono contenuti in componenti biconnessi. Nel figura 3.4 il vertice rosso rappresenta il ladro proprietario della chiave pubblica *pk<sub>red</sub>*, mentre il vertice verde rappresenta la vittima proprietaria della chiave *pk<sub>green</sub>*. Il furto è rappresentato dall'arco verde che unisce la vittima al ladro.

Figura 3.4: Visualizzazione egocentrica del ladro nella rete ausiliaria. Gli archi sono colorati in base al vertice sorgente e la dimensione dei vertici è funzione della loro centralità nella rete.

Guardando tale rete, si nota come ladro e vittima siano collegati a meno del verso da vertici diversi da quello rappresentante il furto. Ad esempio la figura 3.5 evidenzia una sottorete ciclica ricavata dalla rete egocentrica in cui tutti i vertici/indirizzi sono stati contratti in vertici/utente. Si può notare che il furto delle 25000 BTC è

---

<sup>13</sup><http://forum.bitcoin.org/index.php?topic=16457.0>

<sup>14</sup><http://blockchain.info/it/address/1KPTdMb6p7H3YCwsyFqrEmKGmsHqe1Q3jg>

<sup>15</sup><http://mining.bitcoin.cz>

<sup>16</sup><http://blockchain.info/it/address/1J18yk7D353z3gRVcdbS7PV5Q8h5w6oWWG>

<sup>17</sup><http://blockchain.info/it/address/15iUDqk6nLmav3B1xUHPQivDpfMruVsu9f>

---

stato preceduto da un piccolo furto di 1 BTC. Inoltre, sfruttando dati off-network, è stato possibile identificare alcuni degli altri utenti coinvolti: il nodo viola è l'account principale della mining pool Slush e il nodo arancione è il computer appartenente al gruppo hacker noto come LulzSec<sup>18</sup>. Fu fatto anche un tentativo di associare il gruppo hacker al furto tramite la pubblicazione di un comunicato (disponibile in <http://pastebin.com/88nGp508>) in cui l'indirizzo *pk<sub>r</sub>ed* viene indicato come appartenente a LulzSec e utilizzabile per le donazioni al gruppo. Tale tentativo è un falso in quanto il comunicato successivo alla data del furto. Dalla rete si vede comunque come il ladro abbia inviato 0.31337 BTC a LulzSec poco dopo il furto, ma non è possibile associarlo ulteriormente con il gruppo. L'account Slush ha inviato alla vittima un totale di 441.83 BTC in 70 giorni e 0.2 BTC al vertice giallo in un periodo di 2 giorni. Un giorno prima del furto, tale nodo giallo ha effettuato una donazione di 0.120607 BTC a LulzSec. L'utente proprietario del nodo giallo risulta essere proprietario di almeno 5 indirizzi<sup>19</sup> e, come la vittima, è un membro del mining pool Slush e ha donato una volta sola a LulzSec: la sua donazione, il giorno prima del furto, è l'ultima attiva nota di questi indirizzi.

Figura 3.5: Ciclo di transazioni che coinvolge ladro, vittima e altri nodi.

Dato il grande numero di BTC coinvolte, Harrigan e Reid hanno costruito uno strumento che, dato un vertice di partenza, è in grado tracciare flussi consistenti di BTC nel tempo. Applicando tale strumento al furto, è stato creato un elenco di indirizzi che hanno ricevuto alcune delle BTC rubate<sup>20</sup> contenente più di 34100 voci, inoltre è stato possibile individuare una serie di transazioni interessanti. Prima tra tutte è il piccolo furto di 1 BTC che ha preceduto di pochissimo il colpo grosso da 25000 BTC, in secondo luogo la spartizione di tale somma tra un ristretto numero di indirizzi che poi hanno nuovamente confluito il tutto all'indirizzo di partenza. A questa fase di smistamento delle BTC sono seguite quattro interessanti transazioni alle 19:49, 20:01, 20:13 e 20:55, le ultime due delle quali sono particolarmente notevoli in quanto passanti tra un gran numero di indirizzi nel giro di poche ore. Identifichiamo il flusso delle 20:55 come 1 e quello delle 20:13 come 2. Il flusso 1 si divide al vertice A alle 04:05 del giorno dopo il furto, e alcune delle BTC si uniscono al flusso 2 nel vertice B, creando il nuovo flusso 3. Le restanti BTC del flusso 1 passano attraverso numerosi nodi nel corso dei due giorni successivi, creando il flusso 4. Il 16 Giugno 2011 alle 13.37 un piccolo ammontare di BTC passa dal flusso 3 ad un indirizzo mai

---

<sup>18</sup><http://twitter.com/LulzSec/status/76388576832651265>

<sup>19</sup><http://blockchain.info/it/address/1MUpbAY7rjWxvLtUwLkARViqSdzypMgVW4>  
<http://blockchain.info/it/address/13tst9ukW294Q7f6zRJR3VmLq6zp1C68EK>  
<http://blockchain.info/it/address/1DcQvXMD87MaYcFZqHzDZyH3sAv8R5hMZe>  
<http://blockchain.info/it/address/1AEW9TOWWwKoLFYsSLkPqDyHeS2feDVsvZ>  
<http://blockchain.info/it/address/1EWASKF9DLUCgEFqfgrNaHzp3q4oEgjTsF>

<sup>20</sup>elenco completo disponibile nel post del forum che denuncia il furto e in <http://folk.uio.no/vegardno/allinvain-addresses.txt>.

---

visto prima  $pk_1$ <sup>21</sup>. Circa sette minuti più tardi, un altro piccolo numero di BTC viene trasferito dal flusso 3 ad un altro indirizzo  $pk_2$ <sup>22</sup> mai apparso prima. Infine ci sono due transazioni simultanee dal flusso 4 ad altri due nuovi indirizzi  $pk_3$ <sup>23</sup> e  $pk_4$ <sup>24</sup>. Questi quattro indirizzi vengono tutti compressi nello stesso utente  $C$  nella user network. Una rappresentazione di queste transazioni è visibile nella figura 3.6

Figura 3.6: Flussi di transazioni interessati avvenuti in seguito al furto.

Esistono altri flussi interessanti. Ad esempio quello segnato con  $Y$  involve il movimento di BTC attraverso 30 indirizzi mai visti prima in un periodo di tempo molto breve. Ad ogni transazione circa 30 BTC (con un valore di mercato di 500\$ ai tempi) vengono risucchiati dal flusso. Il 20 Giugno 2011 alle 12:35 ognuno di questi indirizzi effettua una transazione ad un indirizzo controllato dal servizio MyBitcoin<sup>25</sup> la quale, coincidenza vuole, era stata precedentemente coinvolta in un altro furto<sup>26</sup>. Questo flusso è visibile in figura 3.7

Figura 3.7: Transazioni molto rapide in seguito al furto.

Tutte queste analisi sono circostanziali. Non è possibile stabilire con certezza se questi flussi implicano una stretta correlazione con i furti, ma sicuramente mettono in luce la quantità di informazioni reperibili e implicano la possibilità che un qualche servizio centrale della rete Bitcoin ne sappia parecchio di più di queste e di altre transazioni.

### 3.4 Contromisure e consigli per aumentare la privacy

**Evitare l'euristica per le transazioni multi-input** Purtroppo non è possibile evitare facilmente questa euristica senza compromettere le operazioni base della rete Bitcoin. Infatti la combinazione di molteplici input permette la creazione di valori più grandi da valori più piccoli: se così non fosse ogni transazione dovrebbe avere un valore minore o uguale ai suoi input, fino a quando non raggiunge il valore minimo di 1 Satoshi. In tale situazione, l'unico modo per combinare gli output di più transazioni sarebbe inviare più transazioni separate formate da un singolo input. Questo oltre ad essere decisamente poco pratico non risolve nemmeno il problema dell'euristica, dato che le transazioni possono comunque essere tracciate da  $\mathcal{A}$  come provenienti dallo stesso portafogli in quanto molto ravvicinate nel tempo. Un'alternativa potrebbe essere modificare il protocollo in modo da permettere a diversi utenti di partecipare

---

<sup>21</sup><http://blockchain.info/it/address/1FKFiCYJSFqxT3zkZntHjfU47SvAzauZXXN>

<sup>22</sup><http://blockchain.info/it/address/1FhYawPhWDvkZCJVBrDfQoo2qC3EuKtb94>

<sup>23</sup><http://blockchain.info/it/address/1MJZZmmSrQZ9NzeQt3hYP76oFC5dWaf2nD>

<sup>24</sup><http://blockchain.info/it/address/12dJo17jcR78Uk1Ak5wfgYXtciU62MzcEc>

<sup>25</sup><http://blockchain.info/it/address/1MAazCWMydsQB5ynYXqSGQDjNQMn3HFmEu>

<sup>26</sup><http://forum.bitcoin.org/index.php?topic=20427.0>

---

ad una stessa transazione: mentre questa soluzione potrebbe effettivamente ridurre drasticamente l'efficacia dell'euristica, pare improbabile che queste transazioni multi-utente possano avere terreno fertile in Bitcoin. Nelle nuove versioni del client ufficiale è però presente la possibilità di creare indirizzi a partire dalla combinazione di più chiavi private invece che da una chiave sola. Gli indirizzi così generati cominciano sempre con il numero 3 invece che il numero 1, e le loro caratteristiche (come il numero di chiavi necessarie e le loro corrispondenti chiavi pubbliche) vengono decise al momento della creazione. Pur non essendo ancora possibili le transazioni multi-input, le transazioni con questi particolari indirizzi possono confondere un attaccante che non è in grado di sapere quante sono le chiavi private coinvolte né se tali chiavi appartengono ad uno stesso utente o a diversi.

**Evitare l'euristica degli indirizzi ombra** Mentre evitare la prima euristica è bene o male fattibile, evitare l'utilizzo degli indirizzi ombra è addirittura controproducente in termini di privacy. Senza l'utilizzo di indirizzi ombra infatti, il resto di una transazione verrebbe semplicemente rimesso nell'indirizzo di partenza della transazione stessa, rendendo ancora più facile tracciarne il percorso. Così come stanno le cose però, sfruttare gli indirizzi ombra causa la dispersione delle monete in diversi indirizzi appartenenti ad uno stesso utente, il che comporta un aggravarsi della falla sfruttabile dall'euristica multi-input. Una possibile alternativa agli indirizzi ombra consiste in un intervento manuale dell'utente: per prima cosa suddividere le proprie BTC nella quantità necessaria per la transazione e poi effettuare la transazione con resto nullo in un momento successivo, abbastanza distante dal momento di suddivisione per non destare l'attenzione degli algoritmi di clustering. Una soluzione che comporta la modifica del protocollo, analogamente a quella proposta in precedenza per le transazioni multi-input, consiste nell'implementazione delle transazioni multi-input multi-output, che renderebbero più complesso distinguere un indirizzo ombra da un indirizzo alla sua prima transazione appartenente ad un altro utente. Esistono alcune possibili implementazioni in tal senso (6.1) ma non sono ancora considerate standard.

**Implicazioni per l'utilizzo generico di Bitcoin** Gli autori di [?], nonostante la specificità della loro simulazione, ritengono che i risultati trovati possano essere applicati anche ad un utilizzo generico della rete Bitcoin. Nello specifico, ritengono che  $\mathcal{A}$  possa estrarre dalla blockchain una piccola porzione di indirizzi che corrispondono ad utenti collocati geograficamente vicino allo stesso  $\mathcal{A}$ , permettendogli quindi di lanciare gli algoritmi di clustering visti in scenari simili a quelli del simulatore. Ad esempio, se Bitcoin fosse diffuso tra i negozianti (alcuni passi in questa direzione si sono fatti negli ultimi mesi in Brasile, Canada e Nord America), allora  $\mathcal{A}$  potrebbe estrarre tutti gli indirizzi che interagiscono con i negozianti presenti in una particolare area geografica: più grande il numero di indirizzi di commercianti (fisici) conosciuti, più completa la visione che  $\mathcal{A}$  ha della rete Bitcoin in quell'area, in quanto si ritiene che solo i clienti più attivi di tali commercianti siano i residenti

---

dell'area. L'unica soluzione a ciò è rappresentata dall'utilizzo di servizi Bitcoin di terze parti, come banche Bitcoin e anonimizzatori, che nascondano la relazione tra gli input e gli output di una transazione. Ovviamente questa soluzione fa letteralmente a pugni con uno dei motivi principali per cui è stata creata la rete Bitcoin, ovvero la rimozione completa delle “terze parti” nelle transazioni. Inoltre si tratta di una relativa *scelta dell'orco*: bisogna rischiare che un  $\mathcal{A}$  tracci il mio profilo oppure devo fidarmi e lasciare in gestione le mie transazioni<sup>27</sup> ad una società che intercede per me, magari anche dietro pagamento?

---

<sup>27</sup>con relativi BTC miei e delle mie controparti



## Capitolo 4

# Sicurezza

### 4.1 Reti P2P in genere

Una rete di computer, di qualsiasi tipo essa sia, richiede necessariamente un certo livello di sicurezza. Questo è ancora più vero per reti pubbliche in cui chiunque può inserirsi, come le reti P2P che si appoggiano a Internet. Tali reti sono un bersaglio particolarmente ghiotto per gli attaccanti proprio a causa della loro maggiore forza: il grande numero di utenti, che equivale ad un grande numero di possibili bersagli. Ogni rete P2P deve quindi confrontarsi con la certezza che alcuni dei suoi nodi siano malevoli. Data la grande varietà di reti P2P, è necessario specificare cosa intendiamo per “reti P2P in genere”. Il modello [?] a cui si farà riferimento consiste nelle seguenti componenti di base:

- Uno spazio degli ID composto da  $b$  bits.
- Un sistema di mappatura degli ID.
- Un sistema di routing, che utilizza una chiave per inoltrare un messaggio alla sua destinazione. Questo include una serie di regole per il churn.

Basandoci sul modello di rete OSI, classifichiamo gli attacchi come di basso, medio e alto livello, a seconda della vicinanza al livello di comunicazione fisico: più il livello dell’attacco è alto, più si basa sul software invece che sull’hardware.

Visto che le reti P2P sono costruite sulla base di altre reti, i livelli ISO/OSI a cui siamo interessati sono il livello applicativo (per il P2P vero e proprio) e il livello di comunicazione (per attacchi basati su TCP e IP).

#### 4.1.1 Attacchi di Basso Livello

##### Denial of Service (DoS)

Uno degli attacchi più basilari che interessa praticamente ogni tipo di rete. Consiste nel bloccare i servizi offerti da uno specifico bersaglio. Si tratta di un attacco estremamente comune esistente in infinite varianti, ma nel caso di P2P la sua versione

---

più semplice consiste nel flood. Consiste nell'estremizzare quanto descritto nel query flood: inondare la rete di pacchetti, legittimi o creati ad-hoc, in modo da ostacolare la normale comunicazione tra i nodi. Semplice, ma estremamente efficace.

Una variante ancora più efficace, e resa tale proprio dalla natura distribuita delle reti P2P, consiste nel **Distribute Denial of Service** (DDoS). Come dice il nome, la differenza sta nel fatto che l'attaccante non è un singolo nodo bensì un insieme di nodi. Il vantaggio in questa tecnica, oltre che nell'immenso numero di pacchetti che inondano la rete, sta nell'anonimato che circonda l'attaccante. Spesso infatti i nodi che a tutti gli effetti inviano i pacchetti sono inconsapevoli di essere parte dell'attacco e manipolati remotamente da un attaccante. Questo aggiunge un ulteriore strato tra l'attaccante e i nodi legittimi che tentano di impedire l'attacco.

Gli attacchi DoS e DDoS diventano tanto più probabili e quanto è vasta la rete, non solo per il maggior numero di nodi compromettibili per un DDoS, ma anche per le politiche aziendali attuate da molte società e provider. Infatti, più la rete è diffusa, più è probabile che firewall aziendali ne limitino l'accesso ai propri utenti e che ISP nazionali ne limitino l'utilizzo. Questo costringe gli utenti a piazzarsi al di fuori di tali reti protette per poter usufruire della rete P2P, e quindi ad esporsi maggiormente agli attacchi.

**Contromisure al DDoS** Il primo problema consiste nell'identificare i DDoS. I sintomi di un DDoS sono del tutto identici a quelli di un normale elevato traffico di rete, in particolare quando i pacchetti usati sono legittimi e non forgiati ad-hoc per l'attacco. Inoltre in un DDoS non tutti i nodi sono stati creati appositamente per attaccare, spesso sono nodi legittimi che vengono utilizzati dall'attaccante per far rimbalzare i suoi pacchetti di attacco. Questi due fattori combinati rendono di fatto *impossibile* bloccare tutti gli attacchi DoS.

Esiste però una tecnica ampiamente diffusa per rendere poco pratico il DoS, o almeno per rallentarlo in maniera drastica. Si tratta del **pricing**, una tecnica che limita la velocità alla quale i nodi possono fare richieste alla rete. Se un nodo deve fare richieste ad un altro nodo (ad esempio, una query per un file), il nodo risponde con richieste di calcoli di hash, esattamente gli stessi calcoli necessari per creare un blocco in Bitcoin. Solo dopo aver risolto il calcolo ed inviata la risposta, la richiesta viene presa in considerazione, tutte le altre comunicazioni inviate nel frattempo vengono scartate, scoraggiando quindi qualsiasi richiesta invadente.

### **Attacco Man-in-the-Middle (MitM)**

Questo attacco consiste nell'inserimento dell'attaccante tra due nodi della rete, in modo che tutte le comunicazioni tra i due nodi passino attraverso l'attaccante. L'attacco è irrilevabile fin tanto che l'attaccante rimane passivo. Una volta ottenute tutte le informazioni che desidera, l'attaccante può diventare attivo modificando i messaggi che vengono scambiati oppure forgiandone di propri spacciandosi per uno o entrambi dei nodi. Inoltre, dato che l'attaccante può influenzare la visione che i

---

due nodi hanno del resto della rete, può creare false identità per simulare messaggi legittimi.

Se questo attacco viene effettuato al livello di rete, l'attaccante è in grado di vedere tutto ciò che passa tra i due nodi e, essendo questo livello inferiore a quello P2P, l'attaccante non ha nessun problema nel creare qualsiasi tipo di pacchetto P2P egli desideri.

Come il DoS ma in misura estremamente maggiore, le reti P2P sono un bersaglio goloso per questo genere di attacchi. Questo perché è difficile inserirsi tra due nodi in una rete normale, ma in una rete P2P è estremamente banale: tali reti infatti non hanno nessun controllo su come sono localizzati i nodi e sono quindi *estremamente* vulnerabili al MitM. Dato che l'attaccante può piazzarsi dove vuole all'interno della rete, gli attacchi risultano essere molto specifici e deterministici, fino anche ad impedire ad un determinato nodo di accedere ad un altro nodo.

### Contrastare il Man-in-the-Middle

Il metodo principale per difendersi dal MitM è rendere tale attacco il più infruttuoso possibile. In una rete senza nodi privilegiati (ad esempio un server centrale, un supernodo o un'autorità centrale di autenticazione), il MitM si limita a compromettere la sicurezza tra due soli nodi nella rete, senza minacciare per nulla il resto dei nodi <sup>1</sup>.

Molte reti (non bitcoin) sfruttano nodi privilegiati per affrontare altre minacce o come parte integrante della loro struttura, e anche nelle reti maggiormente distribuite è possibile effettuare un attacco MitM su larga scala (*Eclipse*).

Il metodo più diffuso per evitare la fuoriuscita di informazioni nella comunicazione tra due nodi è la cifratura a chiave pubblica. Con tale sistema si garantisce l'origine del messaggio, il fatto che non è stato alterato in alcun modo e, volendo, fornisce anche un metodo per evitare che una terza parte non autorizzata possa leggerne il contenuto. L'implementazione di questo semplice meccanismo rende praticamente inutile qualsiasi tentativo di MitM tra due nodi.

#### 4.1.2 Attacchi di Medio Livello

##### Worms

Un worm è un programma auto-replicante simile a un virus che, a differenza di questo, è indipendente da altri programmi presenti sul sistema. La minaccia rappresentata da un worm è decisamente significativa per una rete P2P, in quanto si diffondono a tutti i nodi della rete tramite vulnerabilità presenti ad un livello più basso rispetto a quello della rete stessa. Pur non essendo legati direttamente al P2P, i worm vengono diffusi in modo capillare da esso principalmente a causa di come il P2P viene implementato. In molte architetture infatti i nodi per comunicare tra loro

---

<sup>1</sup>almeno fino a quando la perdita di tre nodi (due vittime e un attaccante) risulta tollerabile per la rete, il che è vero per tutte quelle reti ampiamente diffuse.

---

devono avere installato lo stesso software. Ciò significa che quando questo software ha una certa vulnerabilità (ad esempio, un buffer overflow), tutti i nodi della rete sono vulnerabili. Quindi mentre un worm “normale” deve effettuare una scansione dell’intera rete per trovare degli host vulnerabili, un worm P2P deve solo guardare le tabelle di routing e infettare tutti i nodi vicini, diffondendosi esponenzialmente: in confronto ai worm normali, i worm P2P infettano l’intera rete in modo praticamente istantaneo.

Oltre alla grande velocità di diffusione, il fatto che molte reti P2P siano concepite per il file-sharing e abbiano quindi una grande abbondanza di banda, consente al worm P2P di avere dimensioni maggiori e di essere quindi capace di azioni ed attacchi molto più complicati rispetto ad un worm normale, a volte grande solo come un pacchetto TCP/IP. Molti nodi sono inoltre spesso computer personali di utenti normali che utilizzano quotidianamente Internet. Worm sufficientemente complessi sono in grado di monitorare l’attività dell’utente anche al di fuori della rete P2P e di accedere a dati quali numero di carta di credito, password di account, ecc., rendendo le reti P2P un bersaglio di estremo valore.

Infine, i worm possono usare la rete come uno strumento. Si è parlato prima parlando del DDoS di come un nodo qualsiasi possa diventare un ignaro vettore di attacco. I worm sono il modo in cui questo viene reso possibile: il worm contiene tutto il codice necessario ad effettuare l’attacco, oltre al codice per replicare se stesso negli altri nodi.

**Contrastare i Worm** Il modo principale è mantenere le applicazioni sicure: senza una vulnerabilità comune un worm non può diffondersi in modo efficace. La sicurezza di un software dipende dal modo in cui esso è stato programmato, ad esempio per ridurre il rischio di buffer overflow è possibile usare linguaggi fortemente tipati.

Per ridurre invece l’efficacia di un worm si può decentralizzare il più possibile la rete evitando di implementare nodi privilegiati e/o utilizzare sistemi operativi **hardened**. OpenBSD dalla versione 3.8 per esempio utilizza indirizzi pseudocausali in fase di allocazione della memoria, rendendo quindi difficile sfruttare vulnerabilità presenti nei vari applicativi.

Ma il modo più pratico per difendersi dai worm è mantenere aperta la rete. Ciò significa basarsi su standard diffusi ed aperti per i propri applicativi. Rilasciare i protocolli al pubblico e distribuire il codice dei propri software incoraggia altri sviluppatori ad una analisi critica, il che porta alla più tempestiva scoperta di vulnerabilità ed alla rapida creazione di patch, bug-fix e fork più robusti del software in questione. Inoltre, con opportune licenze, ogni sviluppatore può creare il proprio client per interfacciarsi con la rete costringendo un attaccante a creare un worm specifico per ogni versione di ogni client, e a mantenere tale worm aggiornato mano a mano che nuove vulnerabilità vengono scoperte e risolte o nuovi client implementati. Con una grande varietà di client a disposizione, non tutti i nodi saranno vulnerabili allo stesso identico difetto presente in un altro client.

---

### 4.1.3 Attacchi al livello P2P

#### Comportamento scorretto

Non si tratta di danneggiare una rete intera o un singolo nodo, bensì di trarre il massimo profitto offrendo minima collaborazione. Questo comportamento viene definito genericamente **attacco razionale**. La terminologia si basa sull'assunzione che dietro ogni nodo ci sia un utente razionale che per istinto tenta di trarre il massimo beneficio con il minimo sforzo. Ad esempio nell'ambito della rete BitTorrent un utente scarica un file eliminandolo dalla condivisione non appena completo, oppure riduce ai minimi termini la banda in upload massimizzando quella in download: tale comportamento viene definito significativamente **leeching**, letteralmente *sanguisuga*.

Ci sono molte ragioni per cui un nodo debba comportarsi in questo modo:

- Per preservare la banda in upload, spesso molto limitata dagli ISP.
- Per motivi legali, in special modo dove la condivisione di contenuti protetti dal copyright può risultare in un'azione legale nei confronti del nodo che condivide. Data la natura aperta delle reti, spesso è molto facile risalire all'origine di un contenuto.
- Per istinto: molte persone, se viene lasciata loro possibilità di scelta, tendono a non cooperare per il solo beneficio di aiutare la comunità a cui appartengono, non importa quanto minimo sia il costo che comporta loro.

Come descritto, due sono i metodi in cui ci può implementare questo “attacco”: riducendo le risorse a disposizione della rete oppure riducendo il contenuto condiviso.

**Contrastare i comportamenti scorretti** Ogni rete deve implementare il suo diverso meccanismo per favorire la collaborazione tra i peer: in Bitcoin ci sono le transaction fee, Napster utilizzava un meccanismo di reputazione che favoriva gli utenti che condividevano di più, Samsara (una rete di backup distribuito) consente ad un utente di utilizzare uno spazio su un altro nodo solo pari a quello che l'utente mette a disposizione per gli altri nodi.

Un esempio encomiabile è quello di BitTorrent: il protocollo è disinteressato al numero di file condivisi da un utente o dal contenuto di per se, ma si interessa solamente di quante risorse vengono condivise. Secondo il protocollo BitTorrent i file da condividere vengono suddivisi in parti (**chunks**) di lunghezza variabile che vengono barattati tra i nodi: più un nodo è disposto a dare, più si vedrà restituire. In pratica, più è alta la velocità di upload, più gli altri nodi assegneranno banda a quel nodo, aumentandone la velocità di download.

#### Attacco Sibilla

Si ha questo attacco quando una singola entità malevola rappresenta un grande numero (spesso estremamente elevato) di utenti in una rete P2P con l'obiettivo di

---

assumere il controllo di un segmento della rete. L'attacco si implementa con l'attaccante che tenta di creare un grande numero dei nodi a lui vicini. L'attacco diventa più efficace se l'attaccante è in grado di decidere dove posizionarsi nella rete, in quanto potrebbe aver bisogno di meno nodi per poter causare gravi danni alla rete. Con molti nodi a propria disposizione è possibile controllare tutti i messaggi che passano per il segmento formato dai nodi in questione. Questo attacco è inoltre un **attacco gateway**, il che sta ad indicare una categoria di attacchi solitamente usati come passo preliminare per attacchi su vasta scala di altro tipo, come per esempio l'attacco Eclipse descritto più avanti.

**Contromisure all'attacco Sibilla** La natura aperta delle reti P2P gioca a favore di questo tipo di attacco: senza un'autorità centrale è impossibile fermare completamente un attacco Sibilla. Il meglio che si può fare è renderlo impraticabile.

Per rallentare un attacco si può usare lo stesso metodo usato con gli attacchi DoS: il pricing. Il grande numero di calcoli richiesti per unire molti nodi alla rete può richiedere all'attaccante più tempo di quanto sia disposto ad impiegarne. Se inoltre la rete implementa una sorta di tempo massimo durante il quale un nodo mantiene un certo identificativo, tutti gli attacchi hanno un tempo massimo entro il quale devono essere portati a termine, dopo di che i nodi creati dovranno ricollegarsi alla rete e quindi si dovranno ripetere tutte le pratiche del pricing.

## Attacco Eclipse

L'obiettivo di questo attacco è di separare la rete in due o più partizioni. Quando l'attacco ha successo, tutte le comunicazioni tra le due partizioni devono passare attraverso un singolo nodo malevolo. In pratica questo risulta in un attacco Man-in-the-Middle su vasta scala eseguito a livello applicativo e non a livello di rete, con tutte le potenzialità del MitM normale. Per eseguire l'attacco bisogna piazzare i propri nodi in punti di routing strategici che esistono tra le due partizioni che si vogliono creare. Un attacco Eclipse di successo può distruggere qualsiasi rete P2P, soprattutto quelle che non si curano molto di mantenere tabelle di routing efficienti, perché i nodi finti possono essere posizionati in modo da riempire i vuoti nelle tabelle di routing di ogni altro nodo.

**Contromisure ad Eclipse** Data la similarità di Eclipse con il MitM, le contromisure sono anche simili: cifratura a chiave pubblica. Tuttavia, sebbene MitM non sia una minaccia per la rete, le dimensioni di Eclipse lo rendono estremamente pericoloso anche in caso di cifratura a chiave pubblica. Se i messaggi illeciti indirizzati ai nodi legittimi vengono bloccati, le due partizioni create da Eclipse rimangono di fatto isolate. Con abbastanza nodi piazzati in punti strategici della rete, è possibile creare quante divisioni si desidera, riducendo quindi le dimensioni della rete.

Come per l'attacco Sibilla, è importante impedire ad un attaccante di decidere dove piazzare i suoi nodi. Questo significa che sarà richiesto un elevato numero di

---

nodi per avere una speranza di ottenere sufficiente controllo per poter creare una partizione. Per cui è importante notare che con un attacco Sibilla sufficiente grande è *sempre* possibile eseguire un attacco Eclipse.

#### 4.1.4 Contromisure generali

Si è quindi visto come, oltre alle caratteristiche di tolleranza ai guasti e grande scalabilità che ne rappresentano la base del design, le reti P2P devono anche essere progettate per difendersi dagli attacchi. Solo in questo modo una rete potrà consentire ad ogni nodo di collegarsi e realizzare in pieno il concetto di collaborazione che sta alla base.

In particolare i progettisti di reti P2P dovrebbero implementare le seguenti caratteristiche:

- L'impossibilità per un nodo di decidere in che punto della rete piazzarsi.
- Un limite per la velocità di churn per i nuovi nodi.
- Limitare la velocità di scambio di messaggi tra i nodi, ad esempio con un prining.
- Utilizzare la crittografia a chiave pubblica per garantire solo messaggi legittimi tra i nodi.
- Usare ed implementare solo standard aperti, per diversificare il software a disposizione degli utenti ed irrobustire quelli esistenti.

Se queste caratteristiche vengono implementate, allora tutti gli attacchi fin qui descritti perdono gran parte della loro efficacia, ripagando con la sicurezza il costo che richiede la loro realizzazione.

## 4.2 La rete Bitcoin

### 4.2.1 Fork della blockchain arbitrario

Satoshi analizza questo scenario nel suo whitepaper [?], in cui immagina un attaccante in grado di generare una blockchain alternativa più velocemente di quanto la blockchain ufficiale si evolva. Anche nel caso in cui tale evento si verifichi, la rete non diventa automaticamente vulnerabile ad arbitrarie modifiche da parte dell'attaccante, come creare valore dal nulla o acquisire moneta che non gli è mai appartenuta (attacco doppia-spesa): queste sarebbero transazioni invalide in quanto in conflitto con la blockchain precedente su cui anche la blockchain alternativa si basa, e quindi tali transazioni verrebbero rifiutate da qualsiasi nodo. Un attaccante può solamente tentare di modificare una delle sue transazioni in modo da riappropriarsi dei soldi recentemente spesi. Questa gara che avviene tra la catena ufficiale e il suo fork può

---

essere vista come una passeggiata aleatoria binomiale<sup>2</sup>: l'evento ha successo quando la catena onesta viene estesa di un blocco, aumentando la distanza di +1, mentre il fallimento si ha quando viene creato un blocco per la catena alternativa, riducendo la distanza di -1. La probabilità che un attaccante riduca a zero la distanza partendo da un punto dato della blockchain ufficiale è analoga a quella del problema della *rovina del giocatore d'azzardo*: si immagina che un giocatore d'azzardo con credito illimitato cominci il suo gioco con un dato deficit e giochi un numero potenzialmente infinito di partite per tentare di azzerare il divario. La probabilità che azzeri il divario è la stessa che ha l'attaccante di raggiungere la blockchain onesta, ovvero:

$$q = \begin{cases} 1 & \text{se } p \leq q \\ \left(\frac{q}{p}\right)^z & \text{altrimenti} \end{cases}$$

con

$p$  = probabilità che un nodo onesto trovi il blocco successivo

$q$  = probabilità che l'attaccante trovi il blocco successivo

$z$  = numero di blocchi che separa la testa della blockchain dalla testa della catena dell'attaccante

L'assunzione più plausibile è che  $p > q$ , il che vuol dire che la probabilità diminuisce esponenzialmente tanto maggiore è la distanza da coprire, rendendo la situazione per l'attaccante estremamente precaria.

Il risultato apre però un'altra domanda, ovvero quanto tempo deve aspettare il destinatario di una transazione prima di essere sufficientemente sicuro che tale transazione non possa essere modificata dal mittente. In questo caso, Nakamoto assume che l'attaccante sia appunto un mittente che vuole far credere per un certo tempo al destinatario di essere stato pagato, salvo poi modificare la transazione in modo da indirizzarla a se stesso. Se questo accade il destinatario se ne accorgerà, ma l'attaccante spera che a quel punto sarà troppo tardi. Si assume inoltre che il destinatario della transazione sia un utente accorto, che genera un nuovo indirizzo appositamente per questa transazione. Questa piccola accortezza impedisce al futuro attaccante di precalcolare una fork a suo favore, e quindi di effettuare la transazione solo una volta raggiunto un divario ridotto con la blockchain ufficiale. Quindi l'attaccante deve cominciare i suoi calcoli appena completata la transazione, in parallelo e in segreto. Il destinatario dal canto suo attende che la transazione venga inclusa in un blocco, e che  $z$  blocchi siano stati aggiunti in seguito alla catena. Egli non sa quanto avanti è l'attaccante con i suoi calcoli, ma considerando che i nodi onesti trovino nuovi bloc-

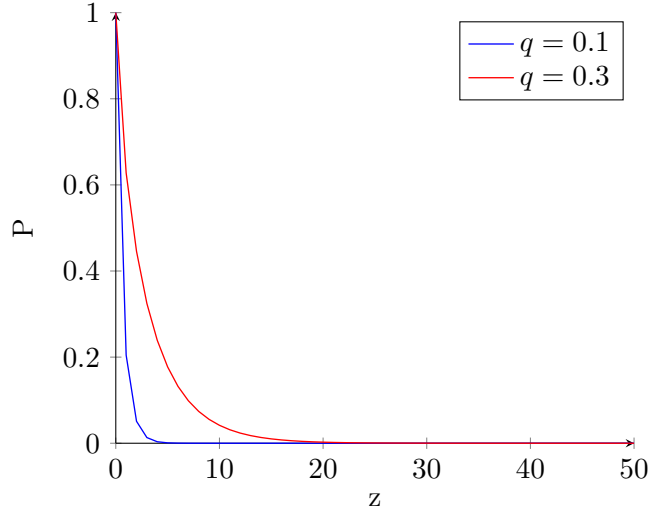
---

<sup>2</sup>Rappresenta formalmente l'idea di procedere in direzioni casuali. Formalmente, una variabile aleatoria discreta  $X(N)$  fornisce, ad esempio, il numero di *testa* usciti dopo  $N$  lanci. Nel caso di 2 possibili scelte, come in questo caso,  $X(N)$  ha una distribuzione binomiale, da cui il nome.



---

Figura 4.1: Probabilità di successo dell'attaccante nella creazione di un nuova blockchain.



chi con la frequenza prevista<sup>3</sup>, il progresso dell'attaccante risulta una distribuzione di Poisson<sup>4</sup> con valore atteso

$$\lambda = z \frac{q}{p}$$

Per ottenere la probabilità che ha ancora l'attaccante di riuscire a raggiungere la blockchain, si moltiplica la densità di Poisson calcolata per ogni possibile progresso compiuto con la probabilità che possa recuperare a partire da quel momento. Tale funzione, una volta modificata in modo da evitare una sommatoria infinita, risulta essere:

$$P = 1 - \sum_{k=0}^z \frac{\lambda^k \cdot e^{-\lambda}}{k!} \left( 1 - \left( \frac{q}{p} \right)^{(z-k)} \right)$$

Un sorgente in C scritto dallo stesso Nakamoto per effettuare questi calcoli è disponibile all'appendice B. Se non bastasse la legge matematica, i risultati ottenuti con tale codice e riassunti dal grafico 4.1 dimostrano come le speranze per l'attaccante diminuiscono esponenzialmente con l'aumentare dei calcoli che deve fare.

---

<sup>3</sup>un blocco ogni 10 minuti secondo le stime iniziali di Nakamoto (2.6

<sup>4</sup>Detta anche *legge degli eventi rari*, esprime il numero di eventi che si verificano consecutivamente ed indipendentemente in un arco di tempo sapendo quanti se ne verificano mediamente in quello stesso intervallo. La funzione di densità di probabilità è  $P(n) = \frac{e^{-\lambda} \cdot \lambda^n}{n!} \forall n \in \mathbb{N}$

---

## History-revision attack

Nonostante i risultati sembrino incoraggianti, secondo quanto riportato in [?] la minaccia risulta essere molto reale, soprattutto a causa della Legge di Moore<sup>5</sup>. Assumendo un numero costante di minatori, la difficoltà per generare un nuovo blocco (che viene dinamicamente calcolata in modo da generare un blocco ogni circa 10 minuti) risulta essere una funzione esponenziale  $f(t) = \alpha e^{\frac{t}{\tau}}$ , da cui risulta che la difficoltà totale dell'intera blockchain in un qualsiasi intervallo di tempo risulta essere l'integrale  $F(T) = \int_{t_0}^t f(t') dt' \propto f(t)$ . Se si pensa ad un attaccante che voglia creare nell'istante  $t_1$  una blockchain con radice nell'istante  $t_0$  alternativa e di difficoltà superiore a quella esistente<sup>6</sup>, allora risulta che, se tale attaccante ha a sua disposizione un piccolo multiplo (ad esempio 2) della potenza di calcolo complessivamente a disposizione dei nodi legittimi, la nuova catena rimpiazzerà quella esistente nell'istante futuro  $t = t_2$ , dove  $\delta t = t_2 - t_1$  è un valore costante che dipende dalla potenza di calcolo (per un multiplo pari a 2 è di 1 o 2 anni). Se questi dati sembrano ancora relativamente sicuri, una piccola analisi della situazione moderna può far cambiare idea. Al giorno d'oggi infatti la maggior parte dei minatori unisce le proprie risorse in *mining pools* (vedi sezione 5) in quanto più conveniente che non creare blocchi in solitario. Un esempio per tutti, la mining pool **deepbit** nel 2012 raccoglieva sotto di sé circa il 40% della potenza di calcolo totale: se avesse raddoppiato la sua quota di mercato, ora come ora potrebbe aver modificato tutte le transazioni mai verificatesi in Bitcoin. Per fortuna gli utente più consapevoli di tale rischio hanno diffuso la voce, e ora non solo **deepbit** rappresenta circa l'1%, ma le due più grandi mining pools, **BTC Guild** e **GHash.IO**, rappresentano rispettivamente il 27% e il 28%, ben lontani dalla soglia di rischio. Un ulteriore fattore di rischio sta nella deflazione insita nel sistema Bitcoin (vedi 7.2), che renderà gli incentivi derivanti dal mining sempre più ridotti, diminuendo quindi il numero di minatori. Di conseguenza la difficoltà per la creazione di nuovi blocchi invece che aumentare comincerà a diminuire e con i nuovi PC che con tutta probabilità esisteranno al tempo, la completa riscrittura dell'intera storia delle transazioni potrebbe diventare un pericolo decisamente concreto.

### 4.2.2 Ridurre il forking: comunicazione tra i nodi

Sebbene le difficoltà che ha un attaccante di realizzare quanto sopra descritto siano minime, esse sono comunque presenti. Ma dato che non si può impedire a qualcuno di ottenere hardware con potenza di calcolo sufficientemente elevata, è necessario trovare altri sistemi per diminuire ulteriormente il rischio di creazione di nuove catene. L'attacco appena descritto è infatti una versione volontaria e premeditata di un evento che si verifica con una probabilità del 1.78% ad ogni blocco trovato, ovvero il forking della blockchain (confrontare la sezione 2.9.2). Chrstian Decker dell'Università di Zurigo e Roger Wattenhofer della Microsoft Research, du-

---

<sup>5</sup>La potenza di calcolo per unità di costo raddoppia ogni anno.

<sup>6</sup>La difficoltà di una blockchain è direttamente correlata con la sua lunghezza: maggiore è la lunghezza, più blocchi saranno stati generati e più è elevata la difficoltà.

---

rante la loro analisi del metodo di propagazione delle informazioni nella rete Bitcoin, dopo aver calcolato la probabilità di forking hanno anche suggerito alcune possibili soluzioni. Secondo i due ricercatori, i forking non volontari si verificano a causa del modo in cui le informazioni si propagano nella rete, per cui le loro soluzioni si basano sul protocollo.

### Ridurre i tempi di verifica

Come si è visto, un fattore importante nel ritardo della propagazione di nuovi blocchi è il tempo necessario per verificare la validità di tali blocchi: più un blocco è grande più tempo richiede la sua verifica<sup>7</sup>. Una prima illuminazione arriva dal fatto che la verifica potrebbe essere divisa in due fasi: un controllo della difficoltà iniziale seguita dalla validazione delle transazioni. Il controllo della difficoltà consiste nel verificare la proof-of-work calcolando l'hash del blocco ricevuto e confrontandolo con l'attuale difficoltà della rete. Viene inoltre verificato che il blocco non sia un duplicato e che faccia riferimento alla testa della catena. La parte difficoltosa sta però nella validazione delle transazioni, che vanno controllate una ad una. La soluzione proposta consiste nell'inoltro del blocco al termine della prima fase, prima di verificare le transazioni, e può quindi essere implementata a livello di client. Ovviamente modificare il comportamento di solo alcuni nodi nella rete potrebbe essere pericoloso ed aiutare un potenziale attaccante. In particolare in questo caso si parla di inoltrare informazioni (blocchi) non completamente verificate, il che potrebbe consentire ad un nodo malevolo di inondare la rete con blocchi falsi e realizzare quindi un attacco DDoS sfruttando le capacità di inoltro di altri blocchi legittimi. Ma, dato che trovare creare un blocco ad-hoc che possa essere inoltrato ha esattamente la stessa difficoltà che creare un blocco legittimo, la soluzione proposta non aumenta il rischio DDoS. D'altro canto però se tale implementazione non è diffusa nella maggior parte dei client (e quindi dei nodi), il vantaggio nella sua adozione è assolutamente marginale.

### Propagare blocchi in pipeline

Un'altra modifica potrebbe essere notificare la presenza di un nuovo blocco con il messaggio *inv* non appena tale messaggio viene ricevuto. Con questo sistema i successivi messaggi *getdata* che richiedono il blocco verrebbero mantenuti in una coda FIFO che verrà evasa a verifica effettuata. Dato che dal solo hash di un blocco non è possibile effettuare alcuna verifica, un nodo malevolo potrebbe annunciare un grande numero di nodi non esistenti, e tali annunci spam verrebbero inoltrati dagli altri nodi nell'intera rete. Una soluzione alla soluzione consiste nel permettere ad un nodo di passare da questa modalità di inoltro-verifica alla modalità verifica-inoltro predefinita, in modo da ridurre la propagazione di messaggi non verificabili<sup>8</sup>.

---

<sup>7</sup>La dimensione massima del blocco, al momento della trattazione dei due ricercatori, era limitata a 500kB, ma attualmente è stata aumentata a 1MB.

<sup>8</sup>Decker e Wattenhofer non specificano però i criteri secondo i quali un nodo dovrebbe cambiare modalità, ma si potrebbe pensare ad un criterio basato sulla frequenza di messaggi non verifica-

---

Ma, anche se si lasciano le cose come descritte, l'invio di messaggi *inv* fasulli non dovrebbe risultare troppo gravoso, dato che tali messaggi hanno una dimensione ridotta e costante di 61B. Inoltre è da fare presente come un attacco simile sia già implementabile tramite la creazione di un numero arbitrario di transazioni e l'annuncio di esse nella rete: fintanto che il nodo attaccante può effettivamente fornire le transazioni su richiesta, non verrà riconosciuto come un attacco. Anche questa modifica agisce a livello di singolo nodo ed è quindi improbabile che possa portare a grandi miglioramenti se applicata solo da un numero ristretto di nodi.

### Aumentare le connessioni

Il problema però più influente è la distanza esistente tra la l'origine di una transazione e i nodi. L'unica soluzione per minimizzare tale distanza è aumentare la connettività dei singoli nodi, incrementando la velocità di propagazione di messaggi *inv*, di blocchi e di transazioni. Ad esempio, portando il numero di connessioni a circa 4000, la distanza media tra due nodi qualsiasi risulta essere circa 2, un valore del tutto ragionevole.

### Test delle precedenti soluzioni

Decker e Wattenhofer hanno implementato le soluzioni da loro teorizzare in un client e le hanno testate sulla blockchain ad altezza compresa tra 200000 e 210000. Durante il test il client è stato connesso ad una media di 3048 nodi e ha caricato circa 20.5 milioni di messaggi *block*. Per ogni blocco il nodo ha ricevuto una media di 2048 richieste. In confronto ai dati rilevati in precedenza (confrontare la sezione 2.9.2), il rateo di forking della blockchain è sceso da 1.69% a 0.78%, ovvero un miglioramento del 53.41%. Come osservato, minimizzare la verifica e il pipelining hanno solo un piccolo effetto su tale miglioramento, ma esso viene moltiplicato dall'aumento delle connessioni. Durante la sperimentazione si è però visto come l'aumento delle connessioni abbia comportato un elevato consumo di banda, con picchi fino a 100MB/s, e ha portato ad un upload totale di circa 2.31TB di dati *block* grezzi.

#### 4.2.3 Impedire riscritture della blockchain con lo scetticismo

Questa riscrittura del meccanismo di accettazione di una blockchain, teorizzato in [?], permette di evitare le gravi conseguenze di attacchi come 4.2.1 basandosi sul principio di senso comune che porta le persone a non fidarsi ciecamente di quanto gli viene detto che contraddice quello che loro già sanno. Tradotto nel mondo bitcoin, vuol dire che un nodo *verificatore* attivo da molto tempo sarà estremamente scettico nell'accettare come legittima una grande modifica della blockchain che altera in modo drastico quella che è la sua visione privata della storia delle transazioni, che si è costruito da solo raccogliendo dati e creando blocchi.

---

bili ricevuti oppure implementare un criterio di coordinazione tra i nodi in modo che una certa percentuale operi sempre nella modalità verifica-inoltro.

---

Il nodo verificatore ha raccolto le sue informazioni memorizzando un timestamp per ogni transazione di cui viene a conoscenza e in privato mantiene periodicamente alcuni snapshot (possibilmente crittografate) della propria visione dello storico delle transazioni. Se in un futuro viene notificata una fork radicale incompatibile con molte delle snapshot salvate, allora prima di accettare la fork deve essere fornita una ulteriore prova della sua validità. Ciò significa che non verranno semplicemente accettate di volta in volta le catene più lunghe, ma prima di ogni accettazione verrà richiesta una ulteriore prova di validità, ovvero una distanza tra i blocchi di testa delle due catene tanto maggiore quanto è la differenza che intercorre tra la nuova catena e la visione privata del nodo.

Si viene quindi a creare un distinguo tra i vari nodi a seconda della loro “anzianità”: i nodi “giovani” che hanno visto poche transazioni e quindi non opporranno molta resistenza a nuove blockchain, e i nodi “anziani”, la cui visione del mondo è molto più difficile da cambiare. Nel caso in cui si dovesse presentare un fork, si assisterebbe quindi ad una momentanea partizione della rete, destinata comunque a risolversi in breve tempo mano a mano che sempre più nodi si schierano e le due catene crescono in lunghezza.

In realtà nel protocollo attuale esiste una sorta di sistema di checkpoint: nel client ufficiale bitcoin sono infatti inseriti a livello di codice alcuni hash di blocchi verificati accuratamente, che vengono eventualmente aggiornati con le nuove versioni del client. Esiste comunque il rischio che il software che si va a scaricare sia una versione compromessa di quella ufficiale, nel qual caso la soluzione proposta sarebbe una efficace contromisura.

- qualcosa sul double-spending

## 4.3 I Portafogli

Come si è visto, la rete Bitcoin può subire alcuni tipi di attacco, ma un utente qualsiasi può fare poco e nulla per aumentare la propria sicurezza in tale ambito. D'altronde, l'utente ha la responsabilità più grande, ovvero mantenere sicuro il proprio patrimonio. Dato che è impossibile per un terzo modificare una transazione che non lo vede come autore, il bersaglio principale di eventuali ladri di Bitcoin sono i punti in cui le BTC vengono “memorizzate”: i portafogli. Dato che una transazione viene firmata con la chiave privata del mittente ed inviata all'indirizzo/chiave pubblica del destinatario, è evidente come la coppia di chiavi sia l'unica cosa che permette di utilizzare le BTC. I portafogli sono le strutture dati che memorizzano, tra le altre informazioni, le coppie di chiavi pubbliche e private necessarie per l'utilizzo delle proprie BTC. Perdere il portafogli vuol dire non poter più usare le monete in esso contenute, esattamente come per un portafogli nella vita reale.

Per questo motivo se si effettua una ricerca online per la sicurezza di Bitcoin, la maggior parte dei risultati riguarderà metodi per rendere più sicuro il proprio

---

portafogli. Il wiki ufficiale di Bitcoin<sup>9</sup> illustra vari metodi per tenere al sicuro il proprio portafogli, dai più semplici a quelli veramente paranoici.

Il consiglio più ovvio è quello di cifrare il proprio portafogli, in modo che sia inaccessibile in caso di furto. La cifratura può essere una funzione implementata dal client oppure realizzabile tramite programmi esterni come ad esempio Truecrypt<sup>10</sup>. La relativa sicurezza<sup>11</sup> così ottenuta ha però diversi svantaggi, tra cui la scomodità, la possibilità di perdere o dimenticare la password e il rischio di corruzione dei dati cifrati.

Un altro consiglio molto diffuso online consiste nell'utilizzare sistemi di *cold storage*, che nel gergo Bitcoin sta a significare un metodo per tenere il proprio portafoglio lontano da Internet (caso in cui si parla di *hot storage*). Un *cold wallet* o *offline wallet* risulterà in un portafogli le cui chiavi private non sono accessibili da nessun client e da nessuna macchina, ma di cui si conoscono le chiavi pubbliche. Sarà quindi possibile unicamente inviare BTC a tali indirizzi, ma non sarà possibile in alcun modo spenderle senza importare le chiavi private in un qualche client. La sicurezza a questo punto si sposta su dove e come conservare le chiavi private. Molti utenti consigliano di non utilizzare dispositivi elettronici, in quanto possono rovinarsi con il tempo, e di fare ricorso invece ai *paper-wallet*. Ogni client genera paper-wallet in maniera differente, a volte includendo anche codici QR per memorizzare le chiavi su dispositivi mobili, ma al lato pratico consistono nella stampa cartacea di stringhe che permettono di recuperare le coppie di chiavi. Mentre per alcuni portafogli le stringhe sono proprio le chiavi, per altri possono essere alcune password che permettono di derivare in modo pseudocasuale le coppie di chiavi. È questo il caso dei portafogli deterministici, in cui tutte le chiavi vengono derivate a partire da un valore iniziale di *seed* che è l'unico valore che serve memorizzare in un eventuale paper-wallet.

Queste tecniche base vengono spesso espanse con criteri di sicurezza aggiuntivi, come l'utilizzo di una distribuzione live durante la generazione dei portafogli e ogni volta che si immettono dati sensibili (per evitare che tali dati vengano memorizzati da qualche parte nel disco rigido oppure "sniffati" da software malevoli eventualmente installati nel sistema), verificare che la stampante non abbia memorizzato informazioni sensibili durante la stampa del paper-wallet, e così via. Esistono anche questioni relative alla sicurezza delle password e dei generatori di numeri casuali, ma la trattazione di tali argomenti esula dall'ambito della tesi.

Naturalmente come sempre accade spetta all'utente decidere il livello di sicurezza di cui necessita e prendere le misure necessarie.

L'implementazione di un particolare tipo di portafogli potrebbe però risultare in un eccellente compromesso tra sicurezza e facilità d'uso: il *super-wallet* proposto in ??, un portafoglio cifrato e distribuito su più dispositivi fisici in modo tale che solo se più autorizzazioni vengono verificate (ad esempio, una password, un file sul filesystem e una chiave privata) si può sfruttare il contenuto. L'utente porterebbe inoltre

---

<sup>9</sup>[https://en.bitcoin.it/wiki/Securing\\_your\\_wallet](https://en.bitcoin.it/wiki/Securing_your_wallet)

<sup>10</sup><http://www.truecrypt.org/>

<sup>11</sup><http://imgs.xkcd.com/comics/security.png>

---

con se una sorta di *sub-wallet*, magari installato nel suo smartphone, configurato con alcune transazioni pre-approvate in modo che si possa prelevare denaro dal portafogli principale in piccole quantità e in un certo intervallo di tempo. Il funzionamento è esattamente identico a quello di una banca che offre servizio bancomat: si può prelevare denaro in quantità predefinite e con una soglia massima mensile. Il vantaggio evidente si ha in caso di furto del portafogli più piccolo: si perde solo quello che vi è contenuto, il resto del proprio denaro è al sicuro in “banca”. Tale portafogli è implementabile con una modifica al protocollo di generazione delle chiavi, ma è retrocompatibile in quanto non richiede modifiche ai meccanismi di verifica.

## 4.4 Bitcoin-Exchange

Esistono però situazioni che non riguardano la rete Bitcoin e che sono al di là delle possibilità di intervento dell’utente: le Bitcoin exchange. Sono questi i servizi in cui un utente può acquistare BTC da un altro utente in cambio di denaro reale e viceversa. Le exchange, al pari di altri servizi esistenti, sono di fatto le uniche autorità centrali presenti all’interno della rete, pur non facendone effettivamente parte a livello di implementazione. I servizi di exchange generalmente richiedono ad un utente iscritto di depositare sul proprio conto online una certa somma in denaro reale oppure in BTC. Una volta depositati, i soldi reali possono essere spesi per acquistare BTC, mentre i BTC possono essere ceduti ad altri utenti che ne fanno richiesta in cambio di soldi reali. Il denaro così guadagnato viene inviato ad un conto precedentemente indicato dall’utente, e solo a questo punto il denaro è effettivamente nelle sue mani. Esiste quindi un periodo in cui i soldi dell’utente sono in mano a terze parti a cui si è costretti a dare totale ed incondizionata fiducia che tale denaro sarà al sicuro per tutta la durata del deposito. Questo, unito al fatto che potrebbe passare molto tempo tra il deposito e l’acquisto<sup>12</sup>

I ricercatori Tyler Moore e Nicolas Christin, dato il numero molto elevato di tali servizi che sono attualmente presenti e l’elevato valore monetario mosso da Bitcoin (a Gennaio 2013 si parla di circa 187 milioni di dollari), hanno deciso di analizzare in modo empirico il rischio che un utente incorre nel dare fiducia a queste exchange(?).

Per la loro analisi, hanno raccolto dal sito <http://www.bitcoincharts.com> dati sui tassi di cambio (che includono anche la quantità di scambi giornalieri e quotazioni medie) fino al 16 Gennaio 2013 per 40 diverse exchange che convertono in 33 diverse valute. Hanno poi calcolato il volume di commercio giornaliero per ogni exchange rapportando il numero totale di Bitcoin convertite in tutte le valute trattate dall’exchange al numero di giorni in cui l’exchange è stata operativa. Tale arco di tempo rappresenta il “lifetime” dell’exchange ed è la differenza tra la prima e l’ultima data in cui è stato effettuato uno scambio. Una exchange viene considerata chiusa se non

---

<sup>12</sup>La compravendita è infatti basata su offerte e ordini in stile borsistico: un utente che desidera acquistare  $X$  BTC piazza un ordine per  $X$  BTC e dichiara che intende pagare  $Y$  denaro reale. Fintanto che un secondo utente non dichiara di voler vendere  $X$  BTC per  $Y$  denaro, l’acquisto non avverrà e l’ordine rimarrà pendente, salvo sua rimozione.

---

è stato effettuato nessuno scambio in almeno due settimane prima del 16 Gennaio 2013. Per le exchange chiuse, sono poi stati passati al setaccio vari forum e siti web che potessero contenere informazioni sulla chiusura in questione, per capire se essa sia stata dovuta ad una breccia nel sistema di sicurezza (che sia un attacco hacker o una frode perpetrata dai titolari dell'exchange stessa). Inoltre sono stati cercati report che indicassero se gli utenti delle exchange chiuse siano stati o meno risarciti per le eventuali perdite subite. Infine ove possibile è stato identificato il paese in cui ha sede l'exchange, per verificare quanto tale paese si sia adeguato alle norme Anti-Money-Laundering and Combating the Financing of Terrorism (AML-CFT) in base ad un indice stilato da alcuni economisti della World Bank<sup>13</sup> che varia da 0 a 49.

I risultati di tale analisi (disponibili per un'analisi dettagliata in ??) mostrano che il rischio di chiusura di una exchange è inversamente correlata con il volume di transazioni<sup>14</sup> che tale exchange gestisce. Di contro, un maggior numero di transazioni rende l'exchange un appetibile bersaglio per gruppi hacker, ed è pertanto direttamente correlato con l'evenienza di attacchi informatici. Si vede quindi che una exchange per operare in modo continuativo ha necessità di effettuare un gran numero di transazioni e contemporaneamente investire in sicurezza per contrastare i sempre più probabili attacchi che si troverà a subire.

---

<sup>13</sup>C. Yepes: Compliance with the AML/CFT international standard: Lessons from a cross-country analysis. IMF Working Papers 11/117, International Monetary Fund (Luglio 2011).

<sup>14</sup>Si intende transazioni nel senso tradizionale del termine, non transazioni bitcoin.



## Capitolo 5

# Mining Pools

### 5.1 Funzionamento generico

Un utente fondamentale per l'ecosistema bitcoin è il miner, ovvero colui che si occupa di cercare nuovi blocchi in modo da rafforzare la catena. Dato che tale ricerca richiede tempo e soprattutto consumo di corrente elettrica a causa dei complicati calcoli crittografici interessati, ogni scoperta di un nuovo blocco viene premiata con un quantitativo di BTC predefinito. Il tipo di attività, il nome di questi nodi e la ricompensa in denaro che ne deriva fanno subito venire in mente un'analogia con i minatori d'oro, che impiegano il loro tempo e il loro duro lavoro per estrarre il prezioso metallo. Mantenendo tale analogia, è evidente come un minatore da solo, pur impegnandosi al massimo delle sue possibilità, non sia in grado di estrarre tanto oro quanto più minatori riuniti in una compagnia mineraria. La controparte Bitcoin delle compagnie minerarie sono le *mining pools*, società private che si prendono il merito del ritrovamento del blocco, ma dividono la ricompensa con quei miners associati che hanno contribuito al ritrovamento. Per sapere quanto conviene associarsi ad una mining pool piuttosto che ad un'altra è necessario fare alcuni calcoli.

Definiamo quindi i seguenti dati:

- B* Ricompensa derivata dal ritrovamento del blocco. Viene dimezzata quando abbastanza blocchi vengono trovati, in modo da limitare il numero massimo di BTC a 21 milioni. Inizialmente era di 50 BTC a blocco, ma al momento della stesura è di 25 BTC.
- D* Difficoltà nel ritrovamento di un blocco, ovvero il numero di bit pari a 0 che devono essere presenti all'inizio dell'hash del blocco perché esso sia considerato valido. Tale valore viene automaticamente aggiustato dalla rete in modo che venga trovato circa un blocco ogni 10 minuti. In pratica tale valore deve essere calcolato in modo che ogni hash abbia una probabilità pari a  $\frac{1}{2^{32D}}$ <sup>1</sup> di essere un blocco valido.

---

<sup>1</sup>Il valore reale di probabilità è  $\frac{2^{16}-1}{2^{48D}}$ , ma come osservato da Meni Rosenfeld in [?], l'approssimazione proposta risulta sufficientemente accurata ai fini dell'analisi.

---

$h$  Hashrate di un nodo, ovvero il numero di hash calcolati nell'intervallo di tempo  $t$ .

Con questi dati presenti, possiamo dire che un singolo miner con hashrate  $h$  nell'intervallo di tempo  $t$  potrà calcolare un totale di  $ht$  hash e quindi una media di  $\frac{ht}{2^{32}D}$  blocchi portandolo ad incassare  $\frac{htB}{2^{32}D}$  BTC.

Ricercare un blocco in solitario, con hashrate fisso, è un processo di Poisson con intensità  $\lambda_r = \frac{h}{2^{32}D}$  (come già accennato in 2.9.2), per cui il numero di blocchi trovati nel tempo rispetta la distribuzione di Poisson con parametro  $\lambda = t\lambda_r = \frac{ht}{2^{32}D}$ . Tale  $\lambda$  risulta essere anche la varianza<sup>3</sup> di questa distribuzione. La varianza del pagamento atteso è  $\lambda B^2 = \frac{htB^2}{2^{32}D}$  e la relativa deviazione standard, espressa come frazione del pagamento atteso, è  $\frac{\sqrt{\lambda B^2}}{\lambda B} = \frac{1}{\sqrt{\lambda}} = \sqrt{\frac{2^{32}D}{ht}}$ .

**Esempio 5.1.** Bob possiede un computer dedicato in grado di calcolare un miliardo di hash al secondo,  $h = 1Gh/s = 10^9h/s$ . Utilizzandolo per un giorno intero ( $t = 86400s$ ) a difficoltà  $D = 1690906$  con ricompensa  $B = 50BTC$ , troverà in media  $\frac{ht}{2^{32}D} \approx 0.0119$  blocchi ricevendo in pagamento  $0.0119B = 0.595$  BTC. Bob si trova però con una varianza nel pagamento di  $0.0119B^2 = 29.75BTC^2$ , con una deviazione standard di  $\sqrt{29.75BTC^2} \approx 5.454BTC$ , ovvero il 917% del guadagno previsto. In effetti, la probabilità che alla fine della sua giornata di lavoro Bob abbia guadagnato qualcosa è solo del  $1 - e^{-\lambda} \approx 1.18\%$ , ben poco per ripagarlo dei soldi investiti nel computer dedicato e della corrente elettrica che esso ha consumato in quella giornata.

Come si nota, la varianza è un valore molto rilevante: un miner con hardware di tutto rispetto, se lavora da solo, potrebbe dover aspettare anche tre mesi prima di ricevere un qualsiasi pagamento, figurarsi per guadagnare effettivamente qualcosa. Inoltre è da notare come il mining sia un processo completamente casuale e privo di memoria: il fatto che siano passati tre mesi senza nessun risultato non rende più probabile il ritrovamento di un blocco, anzi, dovrà aspettare di media altri tre mesi. Se a questo si aggiunge la certezza che la difficoltà è pensata per aumentare nel tempo e di conseguenza anche la varianza, è evidente come diventare un miner solitario è estremamente infruttuoso.

Di contro, associarsi ad una mining pool ha notevoli vantaggi. Una pool infatti in media trova  $\frac{Ht}{2^{32}D}$  blocchi nel tempo  $t$ , con una ricompensa media totale di  $\frac{HtB}{2^{32}D}$ , dove  $H$  è l'hashrate totale a disposizione della pool. Un miner associato il cui hashrate è una frazione  $h = qH$  dell'hashrate totale, dovrebbe ricevere  $q$  volte la ricompensa totale, ovvero  $q\frac{HtB}{2^{32}D} = \frac{htB}{2^{32}D}$  che è esattamente quanto si aspetta di ricevere nel caso in cui abbia lavorato da solo. Il vantaggio sta nella varianza estremamente ridotta:

---

<sup>2</sup> $\lambda_r$  è un parametro di un processo di Poisson detto *intensità* studiato in modo che tale processo rispetti la distribuzione di probabilità di Poisson con parametro  $\lambda = \lambda_r \tau$ , dove  $\tau$  è il tempo trascorso dall'inizio dell'osservazione.

<sup>3</sup>Per un valore casuale, la varianza  $\sigma^2$  è una misura della tendenza che ha tale valore di discostarsi dal suo valore atteso. Di quanto si discosta è stabilito dalla deviazione standard  $\sigma = \sqrt{\sigma^2}$ .

---

la varianza totale è  $\frac{HtB^2}{2^{32}D}$ , per cui la varianza per il singolo è  $q^2 \frac{HtB^2}{2^{32}D} = q \frac{HtB^2}{2^{32}D}$  che corrisponde a  $q$  volte la varianza di un minatore solitario. Il beneficio per un minatore è quindi direttamente proporzionale alla dimensione della pool a cui si associa e inversamente proporzionale alla potenza di calcolo che mette a disposizione della pool.

Una mining pool di solito viene gestita da un operatore il quale prende una tariffa per i servizi offerti, solitamente una percentuale fissa  $f$  sulla ricompensa derivata dal ritrovamento dei blocchi. Quindi per ogni blocco l'operatore si intasca  $fB$  BTC distribuendo  $(1-f)B$  ai minatori, per cui il ricavo atteso di un singolo minatore risulta  $\frac{(1-f)HtB}{2^{32}D}$ . Per stabilire quanto si è contribuito al mining della pool, i minatori trovano e inviano delle fette (*shares*), ovvero hash di un blocco che avrebbero rappresentato un blocco vero se la difficoltà fosse stata pari a 1. Ogni hash calcolato ha una probabilità di  $\frac{1}{2^{32}}$  di essere una share e, assumendo che la funzione di hash sia corretta, dato che è impossibile trovare share senza effettuare gli stessi calcoli necessari per trovare un blocco e anche trovare un blocco senza trovare delle share nel tentativo, il numero delle share trovate da un singolo miner è mediamente proporzionale al numero di hash calcolate dallo stesso miner nel tentativo di trovare un blocco per la pool.

Dato che una share ha una probabilità  $p = \frac{1}{D}$  di essere un blocco valido, se un minatore lavorasse da solo il ritrovamento di una share gli frutterebbe  $pB$ , ammesso e non concesso che riesca a trovare il relativo blocco per primo. Impegnandosi in una pool impiegherebbe le stesse risorse e riceverebbe la stessa quantità per ogni share inviata<sup>4</sup> nel caso in cui un membro qualsiasi della pool trovi il blocco, il che è un evento decisamente più probabile di quello in cui lui da solo trovi il blocco.

La differenza tra le varie pool sta nel modo in cui la ricompensa viene divisa tra i minatori. Infatti se l'hashrate di un minatore è troppo basso, la sua varianza influenza il numero di shares che è in grado di trovare, per cui non tutti i minatori troveranno la stessa pool adatta alle loro esigenze, a causa dei differenti sistemi di retribuzione che sono implementati sulle diverse pool. Inoltre anche il mining discontinuo influenza la varianza in un modo che a sua volta dipende dal sistema di ricompensa. In ogni caso tali considerazioni non hanno effetto sulla quantità del pagamento medio, che risulta sempre essere  $(1-f)pB$  per ogni share inviata.

## 5.2 Sistemi di retribuzione semplici

### 5.2.1 Proporzionale

In questo sistema, i pagamenti sono calcolati in base ad una divisione in *round*, dove un round è il tempo che intercorre tra due ritrovamenti di blocchi da parte della pool. Alla fine di ogni round, quando il blocco viene ritrovato e la pool riceve la relativa ricompensa, l'operatore trattiene la sua parte e distribuisce il resto tra

---

<sup>4</sup> $(1-f)pB$  tenendo conto della tariffa dell'operatore.

---

i minatori in proporzione diretta al numero di share da loro sottoposte durante il round. Se un minatore ha inviato  $n$  share il numero totale di share ricevute dalla pool è  $N$ , allora il pagamento di tale minatore per il round è  $\frac{n}{N}(1-f)B$ . La varianza per ogni share è approssimativamente  $p^2 B^2 \ln D$ , dato che la varianza per un minatore solitario è  $pB^2$  per share, si tratta di un miglioramento di un fattore pari a  $\frac{D}{\ln D}$ . Questo vale unicamente per piccoli miners. Per i miner più importanti la varianza totale non può scendere sotto un valore che è  $q$  volta la varianza del singolo minatore, dove  $q = \frac{h}{H}$  è la frazione di potere computazionale della rete fornito dal minatore. Inoltre tali risultati sono validi solo nel caso in cui il numero di minatori sia costante. Dato che un minatore può essere in grado di stabilire quando è conveniente o meno calcolare blocchi in associazione con una mining pool, è frequente il caso di minatori che si collegano fin quando conviene a loro e si scollegano appena il gioco non vale la candela, pratica detta *pool-hopping* e che permette ai minatori che la sfruttano di guadagnare in quantità superiore alla media per ogni share a discapito di quei minatori che invece contribuiscono in modo continuo alla pool che si troveranno a ricavare meno BTC dal loro lavoro. Nelle pool proporzionali, il pool-hopping funziona particolarmente bene: il pagamento per ogni share, a meno delle tariffe dell'operatore per semplicità, è  $\frac{B}{N}$ , per cui più lungo risulta essere il round, meno si guadagna per ogni share. È quindi evidente come un minatore possa guadagnare di più se invia le proprie share durante i round più corti e cambia pool durante quelli lunghi. Ovviamente nessuno può predire il futuro, ma il passato è ben noto: il numero di share che sono già state inviate nel momento in cui si decide se unirsi o meno ad una pool offre una buona stima di quanto possa ancora essere lungo il round. Ad esempio, se sono già state inviate  $2D$  shares, allora chiaramente entro la fine del round si avrà  $N \geq 2D$  portando il pagamento per ogni share a meno di  $\frac{pB}{2}$ , una situazione decisamente poco remunerativa. Se lavorare per tutto il round porta una ricompensa normale e lavorare solo tardi nel round porta una ricompensa minore, allora è ovvio come lavorare all'inizio del round porti una ricompensa superiore alla media. Secondo Rosenfeld, il punto in cui il pagamento atteso è uguale al pagamento medio si ha quando il numero di share già caricate è il 43.5% della difficoltà: un pool-hopper si sforzerà quindi di collaborare solo prima del raggiungimento di tale valore e per poi abbandonare la pool e magari ritornare in un nuovo round.

Il problema del metodo proporzionale sta nella sua natura deterministica che molto si discosta dalla natura del problema del ritrovamento di un blocco, un processo del tutto casuale. Infatti il numero di share per round segue una distribuzione geometrica (con parametro di successo  $p$  e media aritmetica  $D$ ), la quale è priva di memoria: ci saranno una media di  $D$  shares in ogni round e se un numero  $I$  di tali share è già stato inviato, il numero di share rimanenti non diminuisce, ma segue sempre la stessa distribuzione risultando in mediamente altre  $D$  shares da trovare. Ogni nuova share dovrà competere con le  $I$  precedenti e con le  $D$  future, per cui per migliorare le probabilità di guadagno è importante che l'unico parametro controllabile  $I$  sia il più basso possibile, il che equivale a collegarsi ad una pool all'inizio del round. Una conseguenza del pool-hopping è che i nodi onesti, che rimangono in una pool per

---

tutta la durata del round, possono teoricamente nel caso peggiore guadagnare fino al 43% in meno rispetto al guadagno atteso, una situazione assolutamente inaccettabile che mette bene in evidenza gli svantaggi di un metodo proporzionale.

### 5.2.2 Pay-Per-Share (PPS)

In questo sistema l'operatore non è semplicemente un fornitore di servizi, ma si assorbe tutta la varianza che altrimenti dovrebbero affrontare i minatori: quando uno di esse carica una share, l'operatore immediatamente lo paga con  $(1 - f)pB$  BTC, ovvero con l'intero valore atteso, ma una volta che il blocco viene trovato l'operatore si tiene tutta la ricompensa. Si tratta quindi di un sistema deterministico in cui il guadagno è noto a priori, portando diversi vantaggi per i minatori:

- Nessuna varianza nella ricompensa per singola share. Esiste comunque una varianza nel numero di share trovate, ma è insignificante.
- Nessun ritardo nei tempi di pagamento, in quanto non serve aspettare il ritrovamento del blocco.
- Facile verificare che il pagamento sia avvenuto e che sia nel giusto ammontare.
- Nessuna perdita dovuta a pool-hopping, inefficace con tale sistema.

D'alto canto questi vantaggi sono solo per i minatori, per l'operatore il rischio è invece molto elevato. Come per i minatori nel metodo proporzionale, l'operatore può guadagnare molto in caso di round brevi (in quanto riceve l'intero ammontare della ricompensa pagando solo poche shares) mentre per round lunghi può anche risultare in perdita in modo sostanziale. La sua varianza è infatti la stessa di un minatore solitario il cui hashrate è uguale a quello dell'intera pool. Per compensare tale rischio l'operatore solamente richiede tariffe più alte rispetto agli altri sistemi, e questo è il solo svantaggio a carico dei minatori. A causa dell'elevato rischio, se l'operatore non è oculato nella gestione delle sue finanze la pool può andare in bancarotta. Rosenfeld ha calcolato che per tenere il rischio di bancarotta sotto una certa soglia  $\delta$ , l'operatore deve mantenere una quantità di BTC di riserva almeno pari a

$$R = \frac{B \ln \frac{1}{\delta}}{2f}$$

Tale valore è solitamente più elevato di quanto un novello operatore si aspetta, per cui è importante che egli sia un oculato gestore delle proprie finanze prima di assumersi la responsabilità di aprire un tale tipo di pool. I minatori d'altronde prima di iscriversi a questo tipo di pool dovrebbero verificare le competenze dell'operatore, in modo da non rischiare di perdere i propri guadagni a causa di una bancarotta improvvisa.

---

## 5.3 Sistemi di retribuzione a punteggio

### 5.3.1 Metodo di Slush

Prende il nome dalla pool in cui fu per la prima volta implementato ed è il primo metodo progettato con l'obiettivo di rendere inefficiente il pool-hopping. Si basa sempre sul metodo proporzionale, ma invece di basare la retribuzione sulla semplice conta delle shares, attribuisce uno *score* ad ogni share in base al quale verrà distribuito il premio per il ritrovamento del blocco alla fine del round. Il punteggio è in relazione diretta al tempo trascorso dall'inizio del round, in modo da contrastare l'effetto visto nel metodo proporzionale. La funzione di score è esponenziale:  $s = e^{\frac{T}{C}}$  dove  $s$  è lo score assegnato per una share caricata all'istante  $T$  e  $C$  è una costante. Tale funzione garantisce che ad un certo punto dopo l'inizio del round si raggiunga uno stato stabile in cui non è più rilevante l'istante in cui una share viene condivisa in quanto lo score attribuito risulta essere identico così come di conseguenza la ricompensa. Il parametro  $C$  controlla la velocità con cui si raggiunge tale stato di stabilità o, equivalentemente, la velocità con cui il punteggio decade in relazione alle shares. Se  $C$  è basso il decadimento è rapido, ovvero ogni share ha una elevata possibilità di non ricevere nessun pagamento nel caso di lunghi round. Per round brevi invece il pagamento sarà molto elevato perché non sarà condiviso con molte altre share. Il risultato è che valori bassi di  $C$  aumentando notevolmente la varianza dei pagamenti ricevuti, riducendo però la vulnerabilità al pool-hopping. Nonostante la sua importanza storica, tale metodo ha diversi aspetti negativi:

- Il punto di stabilità viene raggiunto solo ad un certo punto nel round, il che vuol dire che si ripresentano gli stessi problemi del metodo proporzionale relativi a quando associarsi alle mining pools. Sebbene l'effetto non sia altrettanto drastico, la conseguenza è un diretto fallimento delle premesse del metodo che non risulta completamente resistente al pool-hopping.
- Dato che lo score dipende dal tempo e non dal numero di share, il metodo è soggetto ad hopping dovuto a fluttuazioni nell'hashrate.
- Nel calcolo del punteggio non si tiene conto della difficoltà nel trovare un blocco, quindi il metodo può subire hopping derivato dai cambiamenti previsti nella difficoltà.

Il primo problema è il più grave in quanto è un caso di un problema più generale: un sistema che ad ogni divide una ricompensa in modo proporzionale tra i minatori partecipanti al round può essere immune al pool-hopping solo se retribuisce la share che ha risolto il round, il che risulta equivalente al mining in solitario e quindi indesiderabile a causa dell'elevata varianza. Tale affermazione è nota come *teorema dell'immunità da hopping*, dimostrato da Rosenfeld nell'appendice D di [?].

---

### 5.3.2 Metodo Geometrico

Una evoluzione del metodo di Slush che risulta essere immune da hopping. In questo metodo esistono due tipi di tariffe, una fissa e una variabile. La tariffa fissa viene scalata dalla ricompensa di ogni blocco, mentre quella variabile è un punteggio assegnato automaticamente all'operatore all'inizio di ogni round e che decade nel tempo esattamente come il punteggio dei minatori. Questo porta alla creazione di uno stato stabile in cui il punteggio per ogni share rimane costante eliminando le differenze derivanti dai tempi di condivisione delle share. L'algoritmo 3 illustra il funzionamento del metodo.

---

**Algorithm 3** Metodo Geometrico

---

```
Impostare tariffa fissa  $f$ 
Impostare tariffa variabile  $c$ 
for all round do
  contatore per il decadimento:  $s \leftarrow 1$ 
  for all minatore:  $k$  do
    punteggio di  $k$ :  $S_k \leftarrow 0$ 
  end for
  repeat
    if prima iterazione or  $D$  modificata then
       $p = \frac{1}{D}$ 
      tasso di decadimento:  $r = 1 - p + \frac{p}{c}$ 
    end if
    if minatore  $k$  invia share then
       $S_k \leftarrow S_k + spB$ 
       $s \leftarrow sr$ 
    end if
  until blocco trovato
  for all minatore:  $k$  do
    paga  $\frac{(1-f)(r-1)S_k}{sp}$ 
  end for
end for
```

---

Come per gli altri metodi, la tariffa fissa permette all'operatore di intascare  $fB$  sulla ricompensa del blocco, mentre sul rimanente  $(1-f)B$  viene applicata la tariffa variabile permettendogli di guadagnare un ulteriore  $c(1-f)B$ . Complessivamente quindi la tariffa vale in media  $c + f - cf$  e di conseguenza un operatore prende in media  $(c + f - cf)B$  per blocco mentre il restante  $(1-c)(1-f)B$  viene diviso tra i miners a seconda del loro punteggio. Viene utilizzato un contatore dei round  $s$  che verrà utilizzato come multiplo di  $pB$ , in modo da far decadere in modo esponenziale il valore delle prime shares del round mano a mano che questo procede. Il rateo di decadimento è indicato da  $r$ , che imposta il punteggio per ogni share come  $r$  volte il punteggio della share precedente. A difficoltà fissa  $r$  rimane costante in modo

---

che il punteggio dell' $I$ -esima share sia  $r^{I-1}pB$ . Più basso è  $r$  maggiore deve essere il punteggio dell'operatore per mantenere una situazione stabile, per cui maggiore è anche la tariffa variabile. A difficoltà e ricompensa costanti, se  $N$  è il numero complessivo di share sottoposte nel round, allora  $s = r^N$  e il pagamento medio per un minatore risulta essere

$$\frac{(1-f)(r-1)S_k}{sp} = \frac{S_k}{\frac{r^N pB}{r-1}}(1-f)B = \frac{S_k}{\sum_{i=-\infty}^N r^{i-1}pB}(1-f)B$$

che è equivalente a distribuire  $(1-f)B$  in proporzione al punteggio, se l'operatore riceve un punteggio di  $\sum_{i=-\infty}^0 r^{i-1}pB = \frac{pB}{r-1}$  e il punteggio complessivo di tutti i partecipanti è  $\sum_{i=-\infty}^N r^{i-1}pB$ . Lo scopo di ciò è fare in modo di assegnare un numero infinito di share all'operatore all'inizio di ogni round. Così facendo, ogniqualevolta un miner invia una share essa sarà l'ultima di una serie infinita il cui punteggio decade in modo esponenziale nel tempo, permettendo quindi di mantenere fisse le proprietà statistiche della share appena inviata. Dato che il pagamento atteso per ogni share è  $(1-f)(1-c)pB$  la varianza risulta essere approssimativamente  $\frac{(pB)^2}{2c+p}$ , una riduzione di circa  $1 + \frac{2c}{p}$  rispetto alla varianza  $pB^2$  di un minatore solitario. L'operatore si trova in modo simile con una varianza per blocco di circa  $cB^2$ . Se la tariffa media complessiva  $c + f - cf$  rimane costante durante il round a causa della diminuzione di  $f$  e dell'aumento di  $c$ , la varianza per l'operatore viene aumentata mentre viene diminuita quella per i minatori, ma fintanto che  $f$  rimane positivo l'operatore non rischia di subire alcuna perdita alla fine del round. Se però egli sceglie di assorbire maggiore varianza, può rendere negativo  $f$  e aumentare ulteriormente  $c$ , aggiungendo di fatto le sue BTC personali alla ricompensa prevista dal ritrovamento del blocco: ciò significa che può arrivare a perdere fino a  $(-f)B$  per blocco nei round più lunghi, ed impostando  $f = \frac{-c}{1-c}$  la sua tariffa risulta pari a 0. Nel caso limite in cui  $c \rightarrow 0$ , solo la share vincente verrà ricompensata e i minatori stanno a livello pratico lavorando in solitario. Se invece  $c + f - cf$  viene mantenuto fisso con  $c \rightarrow 1$  e  $f \rightarrow -\infty$ , le varianze sopra citate sono trascurabili e il metodo si diventa equivalente al Pay-Per-Share: è quindi consigliato mantenere il valore di  $c$  compreso tra 0 e 1.

Il metodo non è però semplice da mantenere nel caso in cui sia stato implementato in modo approssimativo, soprattutto a causa della crescita esponenziale di  $s$ , per cui è necessario prendere alcune precauzioni:

- Resettare i punteggi: dato che quello che conta è il rapporto tra il punteggio dei lavoratori e il valore corrente di  $s$ , niente viene alterato se tutti questi punteggi vengono divisi per la stessa quantità. Per cui periodicamente è buona norma reinizializzare  $\forall k S_k = \frac{S_k}{s}$  e successivamente  $s = 1$ .
- Usare una scala logaritmica. L'algoritmo 3 memorizza i valori reali di  $s$  e  $S$ , che possono diventare molto elevati. Memorizzando i loro logaritmi  $ls$  ed  $lS$  ed adattando i calcoli si evita l'overflow di tali variabili. Una implementazione in tal senso è visibile nell'algoritmo 4.



---

**Algorithm 4** Metodo Geometrico Logaritmico

---

Impostare tariffa fissa  $f$   
Impostare tariffa variabile  $c$   
**for all** round **do**  
     $ls \leftarrow 0$   
    **for all** minatore:  $k$  **do**  
        punteggio logaritmico di  $k$ :  $lS_k \leftarrow -\infty$   
    **end for**  
    **repeat**  
        **if** prima iterazione **or**  $D$  modificata **then**  
             $p = \frac{1}{D}$   
            tasso di decadimento:  $r = 1 - p + \frac{p}{c}$   
             $lr = \ln r$   
        **end if**  
        **if** minatore  $k$  invia share **then**  
             $lS_k \leftarrow ls + \ln e^{lS_k - ls} + pB$   
             $ls \leftarrow ls + lr$   
        **end if**  
    **until** blocco trovato  
    **for all** minatore:  $k$  **do**  
        paga  $\frac{(1-f)(r-1)e^{lS_k - ls}}{p}$   
    **end for**  
**end for**

---

---

### 5.3.3 Pay-per-lst-N-shares (PPLNS)

Non si tratta di un metodo solo, ma di una famiglia di metodi che si liberano del concetto di round finora visto. Invece di distribuire le ricompense tra i partecipanti di un round, vengono distribuite tra i contributori recenti indipendentemente se sono stati trovati blocchi o meno nel periodo in esame. Così facendo si elimina il concetto di “all’inizio del round” e ne risulta che alcune varianti di PPLNS sono effettivamente immuni al pool-hopping. La variante più semplice consiste nello stabilire un numero fisso  $N$  e pagare una ricompensa di  $\frac{(1-f)B}{N}$  per ognuna delle ultime  $N$  share inviate. Il pagamento per ogni share è  $\frac{(1-f)BL}{N}$  dove  $L$  è il numero di blocchi trovato nelle  $N$  shares. Dato che ogni share ha una probabilità  $p$  di essere un blocco indipendentemente dalle share passate,  $L$  segue la distribuzione di Poisson con media  $\lambda = pN$ , il che porta la ricompensa attesa a  $\frac{(1-f)BpN}{N} = (1-f)pB$ . La varianza di  $L$  è  $pN$  e di conseguenza la varianza per il pagamento atteso è  $\frac{(1-f)^2 pB^2}{N} \approx \frac{pB^2}{N}$ , ovvero  $N$  volte migliore di quella di un minatore solitario. La tempistica dei pagamenti è uniformemente distribuita lungo le  $N$  shares in modo che ogni ricompensa venga ricevuta in media dopo  $\frac{N}{2}$  shares, il che significa che il tempo medio di maturazione delle quote è  $\frac{pN}{2}$ . Aumentare  $N$  riduce la varianza ma aumenta il tempo di maturazione, con l’invariante che il prodotto di questi due dati rimane costante a  $\frac{(pB)^2}{2}$ . Nel caso in cui  $D$  e  $B$  non siano fissi, questa implementazione non è resistente al pool-hopping: i contributi di un minatore sono determinati dalla difficoltà corrente, mentre la sua ricompensa è influenzata dalla difficoltà futura, per cui i pool-hoppers possono utilizzare l’eventuale conoscenza di future fluttuazione nella difficoltà a loro vantaggio, unendosi quando la difficoltà è in prevista diminuzione e andandosene quando è previsto un aumento.

Una variante completamente resistente al pool-hopping è chiamata unit-PPLNS ed è descritta nell’algoritmo 5. In pratica viene tenuta traccia del valore di  $p$  (chiamato *unità*) e di  $B$  (chiamato *amplificatore*) per ogni share nel momento in cui questa viene ricevuta, il numero totale di unità  $U^T$  e  $U_k^T$  ovvero il valore di  $U^T$  nel momento in cui è stata inviata la  $k$ -esima share. Quando un blocco viene trovato e il valore corrente di  $U^T$  viene memorizzato in  $U_n^T$ , per ogni share si ha  $U = U_n^T - U_0^T$  in modo che le uniche share da ricompensare sono quelle per cui  $U_0^T > U_n^T - X - u_n$  dove  $X$  è una finestra temporale in termini di unità e  $u_n$  sono le unità della share che ha trovato il blocco. Di fondamentale importanza è garantire che la struttura dati utilizzata per  $U^T$  e le misurazioni derivate possa supportare la precisione necessaria. Il pagamento totale per ogni share è  $\frac{(1-f)pBL}{X}$  dove  $p$  è l’inverso della difficoltà,  $B$  la ricompensa del blocco al momento ed  $L$  è il numero dei blocchi trovati entro le prossime shares per un totale di  $X$  unità. La probabilità di ogni share di essere un blocco è uguale alle sue unità e di conseguenza  $L$  segue la distribuzione di Poisson con una media  $\lambda = X$  indipendentemente dal passato o da ogni futura modifica della difficoltà. La ricompensa attesa per ogni share risulta essere  $(1-f)pB$ , la varianza approssimativamente  $\frac{(pB)^2}{X}$  (migliorata di  $\frac{X}{p}$  rispetto al minatore solitario), il tempo

---

di maturazione è  $\frac{X}{2}$  e il prodotto tra la varianza e il tempo di maturazione è  $\frac{(pB)^2}{2}$ . La ricompensa attesa per una share come amplificatore  $a$ ,  $u$  unità e un totale di  $U$  unità inviate dopo tale share è  $\frac{(1-f)ua \max(X-U, 0)}{X}$ .

Dato che non esiste nessuna suddivisione in round in questa famiglia di metodi, un'operatore che volesse cambiare i parametri della pool non può farlo tra i round: deve modificarli in ogni share in modo da assicurare che le ricompense attese non cambino.

- Per modificare le unità bisogna procedere in senso inverso dalla prima all'ultima share in modo da non alterare i valori di  $U$ .
- Se si intende modificare  $f$  il metodo più semplice è ridimensionare l'amplificatore di ogni share di un fattore di  $\frac{1-f_1}{1-f_2}$ .
- La modifica di  $X$  si devono riscalarle le unità di ogni share di un fattore di  $\frac{X_2}{X_1}$  e gli amplificatori di  $\frac{X_1}{X_2}$ . In questo modo i valori di  $ua$  e di  $\max(X - U, 0)$  rimangono invariati in ogni share.

Ogni altro metodo di cambiamento (in particolare in caso di riduzione di  $X$ ) potrebbe portare a situazioni in cui per qualche share si ha  $U > X$  e quindi nessun cambiamento nei suoi valori è in grado di mantenere uguale la ricompensa attesa e l'operatore si vede costretto a pagare immediatamente il valore atteso: una sorta di pagamento per il "prestito statistico" che ha sottoscritto quando ha aperto la pool con un valore di  $X$  troppo elevato.

## 5.4 Pay-per-share privo di rischi

L'attrattiva del metodo PPS per i minatori, in contrasto con l'elevato rischio per gli operatori, ha generato molti tentativi di implementazione che mimino il PPS ma con ricompensa di tipo *best-effort* in base alla fortuna della pool.

### 5.4.1 Maximum Pay-Per-Share (MPPS)

In questo metodo vengono mantenuti due bilanci separati per ogni partecipante, un bilancio PPS e un bilancio proporzionale. Ogniqualvolta un minatore invia una share, il suo bilancio PPS viene incrementato come se fosse una pool PPS. Quando viene trovato un blocco, il bilancio proporzionale di ogni minatore viene incrementato come se fosse una pool omonima. Il pagamento che spetta ad ogni partecipante è il minimo tra i due bilanci.

L'idea alla base di tale sistema è, dato che il totale del bilancio proporzionale di tutti i minatori è uguale alla ricompensa totale ottenuta dalla pool e il pagamento che viene elargito ai minatori è minore di questa quantità, la pool non potrà mai andare in perdita. Inoltre, dato che il bilancio PPS di un minatore è un accumulo

---

**Algorithm 5** units-PPLNS

---

```
Impostare tariffa  $f$ 
Impostare dimensione finestra temporale  $X$  {Multipli del tempo medio per trovare
un blocco}
for all round do
  repeat
    if share  $i$  inviata then
       $\text{unit}_i \leftarrow p$ 
       $\text{amplifier}_i \leftarrow B$ 
    end if
  until blocco trovato
   $u_n \leftarrow$  unit della share che ha risolto il blocco
  for all shares  $k$  tranne la risoltrice del blocco do
     $u_k \leftarrow$  unit della share  $k$ 
     $a_k \leftarrow$  amplifier della share  $k$ 
     $U_k \leftarrow \sum_{i \leftarrow k+1} u_i$ 
    paga  $\frac{(1-f)a_k u_k \max\left(\min\left(\frac{X-U+u_n}{u_n}, 1\right), 0\right)}{X}$ 
  end for
end for
```

---

della ricompensa media per ogni share trovata, non può utilizzare il pool-hopping per ottenere una ricompensa più alta di quella media: i round corti non pagano grandi ricompense nell'immediato, bensì riempiono una sorta di buffer del minatore per aiutare i pagamenti durante i round più lunghi. Purtroppo tale metodo ha un clamoroso difetto. Innanzitutto, dato che il totale ammontare pagato è il minimo di due differenti bilanci, ognuno dei quali ha un valore atteso uguale alla media, ne consegue che il pagamento atteso risulta sempre minore alla media: la ricompensa per il minatore è normale in caso di round fortunati e minore del normale in tutti gli altri casi. Rosenfeld ha quantificato che in media la porzione di ricompensa persa risulta essere  $\frac{1}{\sqrt{2\pi n}}$  nel caso in cui un minatore si unisca ad una pool MPPS e collabori per tutto il tempo necessario alla pool per trovare  $n$  blocchi. Mano a mano che il minatore rimane nella pool le perdite relative diminuiscono sensibilmente (nonostante aumentino in senso assoluto), infatti nel caso in cui  $n = 10$  le perdite ammontano a circa il 12.6%.

#### 5.4.2 Shared maximum Pay-Per-Share (SMPPS)

Con questo metodo si tenta di risolvere il problema di MPPS rimpiazzando il buffer per-utente con un buffer globale condiviso tra tutti gli utenti. Round corti riempiono il buffer, aiutando la pool ad elargire i pagamenti PPS durante i round più lunghi. Una volta che il buffer si esaurisce tutti i pagamenti PPS vengono ritardati fin quando non vengono raccolti abbastanza fondi. Come questo viene fatto dipende

---

dall'implementazione specifica della pool. Nella sua concezione originale, la pool tiene traccia della ricompensa spettante ad ogni minatore: quando una share viene inviata la ricompensa viene incrementata come se la pool fosse PPS, mentre per elargire un pagamento esso viene dedotto da tale ammontare. La pool tiene inoltre traccia del suo buffer  $R$ , definito come la differenza tra il totale dei guadagni derivati dal ritrovamento dei blocchi e il totale PPS dovuto ai minatori. Nel caso in cui  $R < 0$ , ovvero esistono partecipanti creditori della pool che non possono essere pagati per mancanza di fondi, il totale dovuto ai minatori è  $-R$ , per cui appena un blocco viene trovato la sua ricompensa viene divisa tra tutti i minatori in proporzione a quanto loro dovuto: un minatore a cui si deve  $w$  si verrà pagato  $\frac{wB}{-R}$ . Il primo inconveniente con questo metodo si ha quando  $R < 0$ : il tempo di maturazione<sup>5</sup> diventa tanto più elevato quanto  $R$  si riduce, essendo approssimabile a  $\frac{-R}{B}$ . Il secondo problema è che sicuramente il buffer prima o poi raggiungerà livelli arbitrariamente bassi ovvero, se la pool continua ad operare indefinitamente, esiste una probabilità del 100% che ad un certo punto il buffer sarà di  $-1000$  BTC, ma anche che in un altro momento sarà  $-10000$  BTC e via dicendo. Quando il debito diventa sufficientemente alto, il tempo di maturazione sarà talmente elevato che probabilmente i minatori non vedranno mai alcun pagamento, abbandoneranno la pool aggravando quindi il problema, in quanto riducendo l'hashrate aumenta il tempo necessario per trovare un blocco e di conseguenza il tempo di maturazione quando misurato in tempo reale. Quando questa spirale distruttiva entra in scena, chiunque abbia un qualche credito con la pool non lo vedrà mai saldato. Un terzo problema è che tale collasso accadrà prima del previsto: inefficienze quali blocchi invalidi, sabotaggi, shares non aggiornate (quando pagate) e via dicendo non fanno altro che ridurre il guadagno atteso da ogni share, provocando un lento ma inesorabile declino del livello del buffer. Infine, questo metodo non è immune da hopping: mentre il pagamento atteso per ogni share è in teoria fisso, il tempo di maturazione non lo è. Lavorare per la pool mentre il buffer è positivo è conveniente in quanto è essenzialmente una PPS senza tariffe, quando il buffer è negativo non è conveniente a causa dell'elevato tempo di maturazione, per cui i minatori tenderanno a lavorare in situazioni di buffer pieno e ad abbandonare la pool altrimenti, lasciando ai minatori onesti tutto il peso del buffer vuoto e invogliando anche essi a lasciare la pool per lidi più fertili.

#### 5.4.3 Equalized shared maximum Pay-Per-Share (ESMPPS)

Si tratta di una variante del metodo precedente basata sull'idea che un seppur minimo pagamento per la share vada elargito appena possibile, mettendo in secondo piano il pagamento completo. In tale metodo, la pool tiene traccia di tutte le share inviate e quanto è dovuto per ognuna di essere. Tali shares vengono pagate in toto fino a quando la pool ha sufficienti fondi, in caso ciò non sia possibile appena viene trovato un blocco la ricompensa viene divisa tra tutte le share in modo da massimizzare il

---

<sup>5</sup>Tempo medio necessario espresso in numero di blocchi per ricevere il pagamento da parte della pool.

---

pagamento minimo elargito ad ognuna di esse (ovvero, si dà la priorità alle share che sono state pagate di meno). Grazie a questo meccanismo, anche se il pagamento dovuto diventa molto grande, le nuove share possono velocemente raggiungere il livello di pagamento di quelle più vecchie. Ma una volta che tale livello di pagamento diventa sufficientemente alto è molto difficile che tale livello aumenti ulteriormente, per cui è molto difficile che la pool saldi al 100% i suoi debiti<sup>6</sup>. Come conseguenza, il tempo di maturazione di questo metodo è di fatto infinito. Il vantaggio di ESMPPS sta nel modo graduale con cui gestisce le perdite. Le inefficienze riducono il livello di pagamento a cui le share si avvicinano, ma il buffer non risulta essere in drastico declino come in SMPPS. Cionondimeno, anche questo buffer raggiungerà sicuramente livelli molto bassi, oltre al fatto che non è resistente al pool-hopping.

## 5.5 Attacchi

### 5.5.1 Pool-Hopping

Una pool vulnerabile a tale attacco possiede caratteristiche di attrattività per il mining, in termini di guadagno atteso, varianza e tempo di maturazione, che variano nel tempo. Il pool-hopping è quindi una tecnica per sfruttare le circostanze favorevoli che consiste nel collegarsi ad una pool quando l'attrattiva è alta ed allontanarsene quando è bassa, portando grandi benefici al pool-hopper a discapito di quei minatori che rimangono collegati anche nei momenti di minore attrattività. Essere costantemente collegati ad una pool vulnerabile è dunque svantaggioso, ma uno scenario in cui il pool-hopping è la normalità è altrettanto insostenibile: prima o poi ogni pool vulnerabile entrerà in uno stato tale da risultare meno attrattiva di una non vulnerabile e quando ciò accade tale pool è destinata a morte certa. L'unico sistema sostenibile è quello in cui ogni minatore opera in solitario o in una pool invulnerabile o quantomeno estremamente resistente al pool-hoppin.

Nella sua forma più tradizionale il pool-hopping si attua sfruttando la ricompensa più elevata elargita per aver minato in round brevi nelle pool proporzionali e nelle pool Slush. Può anche essere attuato nelle pool che utilizzano una qualche forma di buffer: minore il buffer, minore l'attrattiva. Nelle SMPPS i pool-hoppers possono ridurre il loro tempo di maturazione collegandosi solo nei momenti in cui il buffer risulta positivo. In alcune implementazioni ingenuie basate su punteggi che tengono conto di fattori temporanei (e non unicamente su shares e blocchi), l'attacco è reso possibile in base alle fluttuazioni dell'hashrate complessivo della pool. Generalmente è più profittabile collegarsi quando l'hashrate attuale è più alto della media.

---

<sup>6</sup>In effetti, gli operatori di tali pool non considerano il pagamento completo un obiettivo raggiungibile, ma puntano piuttosto a valori attorno al 97%

---

### 5.5.2 Sottrazione di blocchi

Con l'attuale implementazione del protocollo, un minatore è in grado di riconoscere in modo autonomo se una share da lui trovata è o meno un blocco. Un attaccante potrebbe dunque inviare ad una pool solo le share che non sono blocchi per intascare la relativa ricompensa da parte della pool, ma trattenere per se o dilazionare l'invio nel caso in cui la share sia un blocco valido. Questo comportamento può essere usato per portare due tipi di attacchi, sabotaggi e imboscate.

### 5.5.3 Sabotaggi

L'attacco più semplice, in cui l'attaccante non invia mai la share/blocco trovata. In un sistema di ricompensa senza rischio per l'operatore, ogni partecipante, attaccante compreso, perde una parte delle sua ricompensa uguale alla porzione di hashrate dell'attaccante, ovvero in media ogni minatore riceve  $(1 - f) \left(1 - \frac{h}{H}\right) pB$  a share. La perdita può essere significativa, ma dato che è difficile da rilevare potrebbe non causare una fuga dalla pool o altri danni a lungo termine. Se il sistema di ricompensa prevede un rischio per l'operatore, l'intera perdita ricade sulle sue spalle. In media egli guadagna  $\left(f - \frac{h}{H}\right) pB$  per ogni share e, dato che  $f$  è solo una piccola percentuale, il risultato potrebbe benissimo essere negativo, portando la pool alla bancarotta.

### 5.5.4 Imboscata

In questo vantaggioso attacco si ritarda l'invio del blocco appena trovato e si sfrutta la propria conoscenza dell'imminente ritrovamento da parte di altri per focalizzare il proprio lavoro dove più conviene. Se  $T_0 = 10$  min. è il tempo medio necessario per trovare un blocco, si assuma che esistano  $m$  pool PPLNS con parametro finestra  $X = 1$  e hashrate  $H$ , e che l'attaccante lavori contemporaneamente in ogni pool con hashrate  $h$  (*fase di attesa*). Ogniqualvolta trova un blocco, il minatore concentra tutta la sua potenza di calcolo sulla pool in cui lo ha trovato (*imboscata*) e, dopo un certo tempo  $T$  invia il blocco e ricominci a minare in tutte le pool tornando alla fase di attesa. Se un altro blocco è trovato durante la fase di attacco (rendendo quindi il suo blocco invalido), torna alla fase di attesa. Le probabilità di avere successo nell'imboscata (ovvero le probabilità che non venga trovato un altro blocco durante l'intervallo  $T$ ) è  $e^{-\frac{T}{T_0}} \approx 1 - \frac{T}{T_0}$ . Durante l'imboscata troverà in media  $m h T 2^{-32}$  shares ognuna delle quali vale circa  $pB$  più della norma, perché verrà ricompensato sia per il blocco imminente che sta per inviare sia per tutti i blocchi futuri. Comunque, ritardare l'invio del blocco fa decadere le shares esistenti nella pool per un valore totale approssimativamente di  $\frac{pBhT}{H}(H + (m-1)h) \approx pBhT 2^{-32}$ . Nel caso l'imboscata fallisca, non ricaverà nessun extra ma il valore delle shares sarà comunque diminuito, causandogli una perdita di circa  $pBhT 2^{-32}$  dato che il tempo per trovare il prossimo blocco è in media approssimabile a  $\frac{T}{2}$ . A conti fatti, il valore

---

atteso dall'imboscata è dunque

$$pBh2^{-32} \left( \left( 1 - \frac{T}{T_0} \right) (mT - T) - \frac{T}{T_0} \frac{T}{2} \right)$$

Se  $T = \frac{m-1}{2m-1}T_0$  tale valore raggiunge il suo massimo  $pBh2^{-32} \frac{(m-1)^2}{4m-2}T_0 \approx pBh2^{-34}mT_0$ .  
 L'hashrate totale della rete è  $H_0 = \frac{2^{32}D}{T_0}$ , il che porta il valore atteso ad essere equivalente a  $\frac{mhB}{4H_0}$ , dove  $mh$  è l'hashrate dell'attaccante. Ogni share trovata dall'attaccante ha una probabilità  $p$  di essere un blocco valido e rendere fattibile un'imboscata, per cui la ricompensa attesa per ogni share risulta essere  $pB \left( 1 + \frac{mh}{4H_0} \right)$ .

### 5.5.5 Contromisura: Share Offuscate

Una soluzione che prevede la modifica del protocollo Bitcoin consiste nella creazione di share offuscate, ovvero share che, quando trovate dai minatori, non possono essere identificate come blocchi validi. Rosenfeld propone la seguente implementazione per le share offuscate.

- Ogni blocco ha tre campi associati: **SecretSeed**, **ExtraHash** e **SecretHash**. **ExtraHash** è l'hash di **SecretSeed**, sarà parte dell'header del blocco ed uno dei campi utilizzati per calcolare l'hash del blocco. **SecretHash** è l'hash della concatenazione del blocco dell'hash con **SecretSeed**.
- Perché un blocco sia valido, invece di richiedere che il suo hash sia minore di  $\frac{2^{256}}{2^{32}D}$ , si richiede che l'hash del blocco sia minore di  $\frac{2^{256}}{2^{32}}$  e che **SecretHash** sia minore di  $\frac{2^{256}}{D}$ .

L'operatore di una pool sceglierà **SecretSeed** e lo manterrà segreto, calcolerà **ExtraHash** e lo invierà ai minatori insieme agli altri campi che compongono l'hash del blocco. I minatori possono calcolare l'hash del blocco, verificare che sia minore di  $\frac{2^{256}}{2^{32}}$  (il che accade con una probabilità di  $2^{-32}$ ) e in tal caso inviarlo come una share. Non hanno idea se quella share sia un blocco o meno, in quanto non hanno accesso a **SecretSeed** e non possono calcolare **SecretHash**. L'operatore che riceve le share conosce **SecretSeed**, calcola **SecretHash** e se esso è inferiore a  $\frac{2^{256}}{D}$  (probabilità  $p$ ), allora la share è un blocco valido che verrà inoltrato nella rete.



## Capitolo 6

# Scripting

Le transazioni Bitcoin contengono un potenziale nascosto che non viene ancora sfruttato dalle attuali implementazioni del client. Alcuni campi delle transazioni possono infatti contenere piccoli script scritti in un linguaggio di programmazione Forth-like basato su stack valutato da sinistra verso destra, e di fatto nelle implementazioni attuali contengono script per la verifica della correttezza delle transazioni. Il linguaggio di scripting è descritto in [?] ed offre funzioni crittografiche quali **SHA1**, che rimpiazza l'elemento in cima allo stack con il suo hash, e **CHECKSIG** che estrae una chiave pubblica ECDSA e la relativa firma dallo stack, verifica la firma per un messaggio implicitamente definito dai dati della transazione e carica il risultato (vero o falso) nello stack.

Per capire meglio, il seguente listato mostra il contenuto una transazione standard (transazione **pay-to-pubkey-hash**) decodificato dall'esadecimale<sup>1</sup> ed espresso in formato JSON<sup>2</sup>:

```
1 {
2   "hash ":"993830..." ,
3   "ver ":1 ,
4   "vin_sz ":3 ,
5   "vout_sz ":2 ,
6   "lock_time ":0 ,
7   "size ":552 ,
8   "in ":[
9     {
10      "prev_out ":{
```

---

<sup>1</sup>Un esempio di messaggio di transazione non decodificato (così come una chiara rappresentazione di tutte le strutture dati usate dal protocollo) è disponibile in [https://en.bitcoin.it/wiki/Protocol\\_specification#Transaction\\_Verification](https://en.bitcoin.it/wiki/Protocol_specification#Transaction_Verification)

<sup>2</sup>I dati qui sono stati troncati per brevità, ma la transazione è realmente esistente ed è visibile all'indirizzo <http://blockexplorer.com/rawtx/99383066a5140b35b93e8f84ef1d40fd720cc201d2aa51915b6c33616587b94f>

---

```

11         "hash ":"3beabc..." ,
12         "n":0
13     },
14     "scriptSig ":"304402... 04c7d2..."
15 },
16 {
17     "prev_out":{
18         "hash ":"fdae9b..." ,
19         "n":0
20     },
21     "scriptSig ":"304502... 026e15..."
22 },
23 {
24     "prev_out":{
25         "hash ":"20c86b..." ,
26         "n":1
27     },
28     "scriptSig ":"304402... 038a52..."
29 }
30 ],
31 "out":[
32     {
33         "value ":"0.01068000" ,
34         "scriptPubKey ":
35             "OP_DUP OP_HASH160 e8c306... OP_EQUALVERIFY OP_CHECKSIG"
36     },
37     {
38         "value ":"4.00000000" ,
39         "scriptPubKey ":
40             "OP_DUP OP_HASH160 d644e3... OP_EQUALVERIFY OP_CHECKSIG"
41     }
42 ]
43 }

```

I campi della transazione hanno il seguente contenuto:

**hash** è l'hash della transazione calcolato secondo appositi flag (vedere più avanti 6.1) e utilizzato per identificarla.

**ver** indica la versione del protocollo Bitcoin a cui questa transazione si riferisce.

**vin\_sz** è il numero di input presenti.

**vout\_sz** è il numero di output.

---

`lock_time` permette di rendere non pagabile la transazione (e quindi di modificarla) fino ad un certo istante nel futuro. Maggiori dettagli nella sezione 6.1.

`size` la dimensione in byte della transazione.

`in` array degli input.

`prev_out` contiene i riferimenti da cui prelevare le BTC:

`hash` è l'identificativo della transazione a cui fare riferimento.

`n` è l'indice dell'output della transazione di riferimento da cui prelevare le BTC.

`scriptSig` contiene una porzione dello script usato per verificare la transazione. In questo caso è nella forma `<sig> <pubKey>` dove `sig` è firma ECDSA dell'autore della transazione mentre `pubKey` è la corrispondente chiave pubblica.

`out` array degli output.

`value` ammontare di questo output.

`scriptPubKey` seconda porzione dello script. Si trova nella forma `OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG` dove `pubKeyHash` è l'indirizzo del destinatario della transazione mentre le altre voci sono istruzioni del linguaggio di scripting.

La verifica di un input di tale transazione avviene secondo la procedura illustrata nella tabella ??, in cui la transazione da verificare verrà identificata come  $\tau_1$ , mentre la transazione precedente a cui si fa riferimento sarà  $\tau_0$ :

L'altra transazione standard accettata dall'implementazione attuale è quella con cui si pagano le ricompense derivanti dal ritrovamento di un blocco, ovvero le transazioni *coinbase* che appartengono alla tipologia **pay-to-pubkey**. Non essendoci un indirizzo sorgente o una transazione in input, i valori dei campi sono più ridotti: `scriptPubKey` contiene `<pubKey> OP_CHECKSIG` mentre `scriptSig` contiene unicamente `<sig>`. Il funzionamento di questo script è descritto nella tabella ?? e si usano le stesse convenzioni della tabella precedente.

Con il linguaggio di scripting è anche possibile effettuare alcune transazioni particolari non ancora considerate standard, per realizzare le quali è necessario utilizzare dei clienti appositi in grado di manipolare la struttura grezza delle transazioni. Ad esempio è possibile creare una transazione il cui output non è spendibile semplicemente inserendo all'inizio di `scriptPubKey` il comando `OP_RETURN`, il quale termina subito l'esecuzione dello script rendendo la transazione non utilizzabile in un futuro input in quanto il contenuto di `scriptSig` diventa inutile. Esiste un esempio di tale tipo di transazione<sup>3</sup> che avendo l'unico output impostato a 0, lascia l'intero ammontare di 0.125 BTC come transaction fee al minatore che troverà il blocco.

---

<sup>3</sup><https://blockexplorer.com/tx/eb31ca1a4cbd97c2770983164d7560d2d03276ae1aee26f12d7c2c6424252f29>

Stack	Script	Descrizione
	$\langle \text{sig} \rangle$ $\langle \text{pubKey} \rangle$ OP_DUP    OP_HASH160 $\langle \text{pubKeyHash} \rangle$ OP_EQUALVERIFY OP_CHECKSIG	Unisco il campo <code>scriptSig</code> di $\tau_1$ con il campo <code>scriptPubKey</code> del relativo output di $\tau_0$ creando lo script da eseguire
$\langle \text{sig} \rangle$ $\langle \text{pubKey} \rangle$	OP_DUP    OP_HASH160 $\langle \text{pubKeyHash} \rangle$ OP_EQUALVERIFY OP_CHECKSIG	Carico nello stack la firma dell'autore di $\tau_1$ e la relativa chiave pubblica
$\langle \text{sig} \rangle$ $\langle \text{pubKey} \rangle$ $\langle \text{pubKey} \rangle$	OP_HASH160 $\langle \text{pubKeyHash} \rangle$ OP_EQUALVERIFY OP_CHECKSIG	Duplico la chiave pubblica in cima allo stack. Si dovrà utilizzare due volte: prima per verificare che l'autore di $\tau_1$ sia il destinatario di $\tau_0$ , e poi per verificare che possieda la chiave privata necessaria per spendere le BTC
$\langle \text{sig} \rangle$ $\langle \text{pubKey} \rangle$ $\langle \text{pubHashA} \rangle$	$\langle \text{pubKeyHash} \rangle$ OP_EQUALVERIFY OP_CHECKSIG	Calcolo l'hash della chiave pubblica dell'autore di $\tau_1$ in cima allo stack e lo carico nello stack stesso
$\langle \text{sig} \rangle$ $\langle \text{pubKey} \rangle$ $\langle \text{pubHashA} \rangle$ $\langle \text{pubKeyHash} \rangle$	OP_EQUALVERIFY OP_CHECKSIG	Carico nello stack la chiave pubblica del destinatario di $\tau_0$
$\langle \text{sig} \rangle$ $\langle \text{pubKey} \rangle$	OP_CHECKSIG	Verifico che i due elementi in cima allo stack siano uguali, ovvero che l'autore di $\tau_1$ sia anche il destinatario di $\tau_0$
true		Verifico la firma (ovvero il possesso da parte dell'autore di $\tau_1$ della necessaria chiave privata) e salvo il risultato nello stack

Tabella 6.1: Passaggi della verifica dell'input di una transazione standard tra due utenti.

Stack	Script	Descrizione
	<code>&lt;sig&gt;      &lt;pubKey&gt; OP_CHECKSIG</code>	Unisco <code>scriptSig</code> da $\tau_{u_1}$ e <code>scriptPubKey</code> da $\tau_{u_0}$ creando lo script da analizzare
<code>&lt;sig&gt; &lt;pubKey&gt;</code>	<code>OP_CHECKSIG</code>	Carico nello stack la firma dell'autore di $\tau_{u_1}$ e la chiave pubblica del destinatario di $\tau_{u_0}$
<code>true</code>		Verifico la firma e salvo il risultato nello stack

Tabella 6.2: Passaggi della verifica di una transazione standard di generazione di BTC.

Stack	Script	Descrizione
	<code>&lt;data&gt;    OP_HASH256 &lt;given_hash&gt; OP_EQUAL</code>	Unisco <code>scriptSig</code> e <code>scriptPubKey</code> creando lo script da analizzare
<code>&lt;data&gt;</code>	<code>OP_HASH256 &lt;given_hash&gt; OP_EQUAL</code>	Carico nello stack <code>scriptSig</code>
<code>&lt;data_hash&gt;</code>	<code>&lt;given_hash&gt; OP_EQUAL</code>	Calcolo l'hash del dato
<code>&lt;data_hash&gt; &lt;given_hash&gt;</code>	<code>OP_EQUAL</code>	Carico nello stack l'hash di confronto
<code>true</code>		Confronto i due hash

Tabella 6.3: Passaggi della verifica di una transazione di esempio contenente un puzzle.

Dal lato opposto, è anche possibile creare transazioni spendibili da chiunque<sup>4</sup> lasciando vuoto `scriptPubKey` e inserendo `OP_TRUE` in `scriptSig`.

Un ulteriore esempio delle potenzialità dello scripting è la creazione di alcuni puzzle crittografici che devono essere risolti per poter spendere le BTC. Esiste un esempio<sup>5</sup> in cui si richiede di trovare qualche dato tale che, sottoposto due volte ad hash, risulti uguale ad un dato fornito. La tabella ?? illustra l'esecuzione di un simile script.

<sup>4</sup>Un motivo per utilizzare in futuro questo tipo di transazioni è illustrato nel Wiki all'indirizzo [https://en.bitcoin.it/wiki/Fidelity\\_bonds](https://en.bitcoin.it/wiki/Fidelity_bonds).

<sup>5</sup><https://blockexplorer.com/tx/a4bfa8ab6435ae5f25dae9d89e4eb67dfa94283ca751f393c1ddc5a837bbc31b>

---

## 6.1 Contratti

I contratti distribuiti sono un modo per sfruttare le potenzialità dello scripting Bitcoin per creare accordi tra persone in modo da limitare al minimo la fiducia da riporre in esse. Come per alcuni degli script precedentemente descritti, anche questi non sono considerati standard e sono anzi da considerare solo come esempi di una possibile futura evoluzione delle rete che permette nuove forme di transazioni evadibili in automatico tramite la blockchain senza necessità di intervento umano. Gli esempi sono presi dal Wiki ufficiale ([?]) che a sua volta prende spunto da una dissertazione di Nick Szabó ([?]).

I contratti vengono realizzati sfruttando alcune caratteristiche delle transazioni e del linguaggio di scripting che non sono considerate ancora standard. Una di queste caratteristiche è il *time-lock* visto in precedenza: ogni transazione può infatti possedere un blocco temporale che ne consente la modifica e l'eliminazione entro una certa tempo (ad esempio un timestamp o l'indice di un blocco) e se impedisce lo sfruttamento dei BTC coinvolti fino a quel momento. Una transazione con time-lock che ha superato l'istante previsto diventa definitiva. Nel caso di modifica di una transazione viene incrementato un contatore interno (uno per ogni voce di input della transazione) fino ad un valore massimo. Per cui anche nel caso in cui il time-lock non sia stato raggiunto, se tutti i contatori sono arrivati al loro valore massimo (definito come `UINT_MAX`), allora la transazione viene considerata definitiva. I contatori possono essere utilizzati per generare nuove versioni di una transazione senza invalidare le firme precedenti, ad esempio nel caso in cui si voglia aggiungere un input proveniente da un diverso utente mediante l'utilizzo dei flag precedentemente accennati. Tali flag **SIGHASH** definiscono il modo in cui viene creato l'hash che viene utilizzato nella fase di verifica della firma e permettono dunque di costruire transazioni multi-utente in cui ciascuno degli utenti coinvolti firma una parte della transazione senza dover (e poter) firmare o influenzare le altre parti.

I flag sono composti da due componenti, una modalità e un modificatore. Le modalità possibili sono 3:

**SIGHASH\_ALL:** è il valore di default, indica che tutta la transazione viene firmata ad eccezione degli script di input (che se venissero firmati anch'essi renderebbero impossibile la verifica, per cui sono sempre esclusi dalla firma). Vengono quindi incluse nella firma le altre proprietà dell'input quali gli output connessi e i contatori. È equivalente a firmare un contratto che reciti "Sono d'accordo di impiegare danaro proveniente da questi miei fondi se tutti impiegano il danaro proveniente dai loro fondi e il totale impiegato è quello definito."

**SIGHASH\_NONE:** gli output non vengono firmati e possono contenere qualsiasi cosa. Il contratto sarebbe "Sono d'accordo di impiegare danaro proveniente da questi miei fondi fintanto che tutti impiegano il danaro proveniente dai loro fondi, ma non sono interessato a come verrà impiegata tale somma o se verrà usata nella

---

sua totalità.”. Con questa modalità si autorizza altri utenti ad aggiornare la transazione modificando i contatori relativi ai propri input.

**SIGHASH\_SINGLE:** come per la modalità precedente, gli input sono firmati ma non i contatori, in modo che altri utenti possano aggiornare la transazione. L’unico output che viene firmato è quello inserito nella transazione nella stessa posizione dell’input<sup>6</sup>. Il contratto equivalente è “Accetto il contratto fintanto che il mio output sia quanto io ho stabilito, indipendentemente dagli altri firmatari.”.

Il modificatore è invece **SIGHASH\_ANYONECANPAY**, che, se inserito in uno script di input, indica che tale input viene firmato ma non è necessario che lo siano anche gli altri. È inoltre possibile utilizzare più chiavi pubbliche tramite l’opcode **CHECKMULTISIGVERIFY** tramite il quale si possono fornire più chiavi pubbliche e richiedere un numero minimo di firme che devono essere valide, numero che può anche essere minore del numero di chiavi fornite. Ad esempio un output può richiedere due firme per essere verificato con lo script `2 <pubKey1> pubKey2 2 CHECKMULTISIGVERIFY.` Infine un aspetto fondamentale per realizzare un contratto è la tecnica di propagazione nella rete dello stesso: per essere valido (in termini legali, non in termini crittografici), il contratto deve essere visibile a tutti i possibili firmatari prima della firma, in modo che essi sappiano quale accordo stanno sottoscrivendo. Esistono due metodi generici per fare ciò in modo sicuro (ovvero con minima fiducia tra le parti):

- Inviare le transazioni tra i possibili firmatari al di fuori della rete Bitcoin in forma incompleta o invalida e inviato una volta completo.
- Utilizzare due transazioni. La prima (il contratto) viene creata e firmata ma non inoltrata nella rete. L’altra transazione (il pagamento) viene inoltrata dopo che il contratto è stato siglato in modo da bloccare le BTC, dopodiché viene inoltrato il contratto.

Tutti questi strumenti combinati possono portare alla creazione di nuovi strumenti finanziari che si basano sulla rete Bitcoin, come ad esempio quelli illustrati brevemente di seguito. Per maggiori dettagli o altri possibili contratti sono disponibili informazioni più approfondite in ?? e in ??.

### 6.1.1 Deposito temporaneo

Rimanendo nell’ambito della rete Internet, è possibile voler sottoscrivere un account presso un sito Internet che richieda una certa fiducia nell’utente, ad esempio per verificare che egli non sia uno spambot. Una soluzione possibile consiste nell’inviare una caparra all’operatore del sito, fidandosi del fatto che tale somma verrà restituita in caso di cancellazione dell’account e che l’operatore non scapperà con il denaro. Tale fiducia può essere eliminata con un contratto Bitcoin secondo la seguente procedura:

---

<sup>6</sup>Per chiarezza, tornando all’array degli input e degli output in formato JSON, l’output all’indice  $x$  verrà firmato tramite l’input all’indice  $x$

- 
1. Utente ed operatore si scambiano una coppia di chiavi pubbliche appositamente create.
  2. L'utente crea la transazione  $\tau_1$  (il pagamento) contenente X BTC in un output che richiede la firma sia dell'utente che dell'operatore, ma non la inserisce in rete.
  3. L'hash della transazione appena creata viene inviata all'operatore.
  4. L'operatore crea una seconda transazione  $\tau_2$  (il contratto) la quale paga l'output di  $\tau_1$  all'utente. `lock_time` viene impostato ad una qualche data nel futuro il contatore viene inizializzato a 0. Dato che l'output di  $\tau_1$  richiede la firma sia dell'operatore che dell'utente per essere speso, la transazione è incompleta.
  5.  $\tau_2$  incompleta viene inviata all'utente, il quale verifica che il denaro prima o poi tornerà nelle sue tasche alla data designata, salvo modifiche. Essendo il contatore impostato a 0, il contratto può essere modificato in futuro se entrambe le parti sono concordi. Lo script in input di  $\tau_2$  è incompleto visto che manca la firma dell'utente, che verrà apposta una volta verificato il contratto, completando così la transazione.
  6. L'utente può ora inviare  $\tau_1$  per pagare l'operatore con la somma pattuita e successivamente inoltrare  $\tau_2$  per rendere valido il contratto.

Una volta completati tali passaggi, le X BTC inviate con  $\tau_1$  si trovano in uno stato in cui non possono essere spese né dall'utente né dall'operatore senza l'approvazione dell'altro. Allo scadere del `time_lock` il contratto si concluderà e l'utente riavrà indietro le sue monete anche nel caso in cui il sito o l'operatore siano spariti.

Nel caso in cui l'utente voglia chiudere anticipatamente il contratto, l'operatore crea una nuova versione di  $\tau_2$ , ad esempio  $\tau_{2.1}$ , in cui `lock_time` è impostato a 0 e il contatore al suo massimo `UINT_MAX` e applica la sua firma.  $\tau_{2.1}$  viene inviata all'utente, il quale controlla, eventualmente firma e inoltra la transazione alla rete invalidando  $\tau_2$  e riappropriandosi subito delle sue BTC.

La stessa procedura viene applicata se l'utente volesse prolungare il contratto: basta impostare un nuovo `lock_time`, incrementare di una unità il contatore, firmare ed inoltrare.

### 6.1.2 Acquisto di beni con mediatore

La preoccupazione più grande di tutti coloro che acquistano online o con altri mezzi telematici è “Il prodotto che ricevo (se lo ricevo) sarà quello per cui ho pagato? E se non sono soddisfatto riavrò i soldi indietro?”. Simile è anche la preoccupazione di chi vende nel caso in cui il pagamento venga effettuato dopo la spedizione o dopo aver dovuto effettuare alcune spese: “Riceverò mai il denaro che mi spetta?”. Sfruttando piattaforme quali eBay o Amazon è possibile richiedere l'intervento di un mediatore



---

(appunto, eBay o Amazon) per risolvere la disputa e assegnare il denaro a colui che ne ha diritto.

Sfruttando Bitcoin e le opzioni di firma multipla viste è possibile eseguire le stesse operazioni, ovvero creare un pagamento tra tre parti che rimane bloccato fintanto che due delle tre parti non decidono a chi spetta il denaro. La procedura è la seguente:

1. Ognuna delle parti crea la sua chiave pubblica:  $k_c$  per il commerciante,  $k_a$  per l'acquirente e  $k_m$  per il mediatore. Commerciante e mediatore inviano le proprie chiavi all'acquirente.
2. L'acquirente invia al commerciante  $k_m$ .
3. Il commerciante "sfida" il mediatore con una nonce casuale, che viene da lui firmata con la porzione privata di  $k_m$ , garantendo che tale nonce appartiene proprio al commerciante.
4. L'acquirente crea la transazione  $\tau_1$  nel cui output sarà presente lo script `2 <Kc> Km Ka 3 CHECKMULTISIGVERIFY` e la inoltra nella rete.

La situazione che si viene a creare blocca le monete inviate con la transazione in uno stato di inspendibilità fino a quando non avviene uno dei seguenti fatti:

- Commerciante ed acquirente raggiungono un accordo (il commercio è andato a buon fine oppure il commerciante accetta di rimborsare il cliente).
- Acquirente e mediatore si accordano (il commercio è andato male e il mediatore dà ragione all'acquirente).
- Il mediatore spalleggia il commerciante (i beni sono comunque arrivati a destinazione come richiesto e il denaro spetta al commerciante).

Infatti l'output della transazione inoltrata, per essere speso, richiede che l'input della nuova transazione contenga almeno due delle firme collegate alle chiavi pubbliche elencate. Per cui una volta raggiunto l'accordo, una delle parti crea una  $\tau_2$  con `scriptSig` contenente la sua firma, la inoltra alla seconda parte interessata la quale la firma e la inoltra alla rete assegnando il denaro a chi di dovere.

### 6.1.3 Raccolta fondi assicurata

---

<sup>7</sup>Mia interpretazione di "Assurance Contract" di cui non trovo la traduzione vera.



## Capitolo 7

# Argomenti correlati

### 7.1 Statistiche

Le tecniche descritte in 3.1.2 sono state utilizzate dai ricercatori Dorit Ron e Adi Shamir del Weizmann Institute of Science di Israele per condurre alcune analisi statistiche sulla rete delle transazioni Bitcoin dalla prima transazione del 3 Gennaio 2009 al 13 Maggio 2012. I risultati da loro trovati verranno qui esposti in forma sintetica sotto forma di punti solo come nota informativa di alcune proprietà della rete, in quanto la trattazione è ormai vecchia di quasi 2 anni, durante i quali il mondo Bitcoin si è espanso esponenzialmente, minando la corrispondenza di tali statistiche con l'attuale realtà. I risultati completi sono disponibili in [?]. Statistiche aggiornate sono invece disponibili in tempo reale presso [?].

Prima di esporre i risultati è importante specificare alcune osservazioni fatte dagli stessi osservatori. Come descritto in 3.1.2, il grafo delle transazioni viene chiamato *grafo degli indirizzi* mentre il grafo derivato dall'unificazione degli indirizzi prende il nome di *grafo delle entità*, il quale è per forza di cose un risultato approssimato e potrebbe essere vittima di due stime errate. È possibile infatti che si sia verificata una sottostima nell'assegnazione di un indirizzo ad una entità nel caso in cui non ci siano abbastanza dati (transazioni) per associare tale indirizzo ad una entità, ma è anche possibile sovrastimare il numero di entità nel caso in cui molti utenti abbiano deciso di unificare le loro attività in una unica transazione a cui ciascuno di essi contribuisce con il proprio indirizzo. Avendone parlato con gli stessi sviluppatori Bitcoin, i ricercatori hanno stabilito che gli errori di sovrastima sono estremamente ridotti, ma potrebbero essere stati commessi alcune sottostime, come già rilevato in 3.1.2.

I risultati più rilevanti della trattazione sono i seguenti:

- I dati analizzati contano 3730218 indirizzi distinti, dei quali 609270 compaiono solo come destinatari di transazioni.

- 
- I 3120948 indirizzi che inviano BTC sono stati aggregati in 1851544 entità distinte, che unite agli indirizzi unicamente destinatari, porta il totale delle entità a 2460814.
  - Esiste una varianza notevole nelle statistiche a causa di una sola entità che possiede 156722 indirizzi. L'entità in questione è l'exchange Mt. Gox, responsabile all'epoca di circa il 90% del totale degli acquisti di BTC.
  - Dato che il bilancio medio di un indirizzo è di 2.4 BTC, e quello di una entità è 3.7 BTC, a causa degli errori sopra descritti si stima che il bilancio medio di un utente sia superiore alle 3.7 BTC.
  - All'epoca dell'analisi esistevano 9000050 BTC, generati dai 180001 blocchi analizzati, ma sommando i bilanci degli indirizzi che hanno solo ricevuto BTC si vede che essi contengono 7019100 BTC, ovvero il 78% del totale.
    - Di queste, il 76.5% (il 59.7% del totale) sono *vecchie BTC*, nel senso che sono state ricevute più di tre mesi prima del 13 Maggio 2012 e che non sono mai state spese fino a tale data.
    - Alcune di queste monete potrebbero essere smarrite o abbandonate da utenti che muovevano i loro primi passi nel mondo ancora acerbo di Bitcoin, e non per forza BTC accumulate per avidità.
    - Per precauzione Dorit e Adi hanno deciso di ignorare tutte le transazioni precedenti il 18 Luglio 2010, data di apertura di Mt. Gox, portando il numero di monete *sparite* dalla rete a 1657480.
    - Ripetendo i calcoli, il 73% di tutte le BTC rimanenti è depositato in indirizzi dormienti, e il 70% di tali indirizzi (il 51% del totale) risultano vecchie.
  - Calcolando in base agli indirizzi invece che in base alle transazioni, risulta che il 55% di tutte le BTC è dormiente, dato non affetto da eventuali errori di stima precedentemente descritti.
  - Il numero totale di BTC coinvolte in transazioni fino al 13 Maggio 2012 è di 423287950 (mining escluso).
  - Il 52% delle entità (59% degli indirizzi) hanno ricevuto meno di 10 BTC, e l'88% delle entità (91% degli indirizzi) meno di 100 BTC.
  - 76 entità (129 indirizzi) hanno ricevuto tra le 400000 e le 800000 BTC e solo quattro entità (un indirizzo) hanno ricevuto più di 800000 BTC.
  - Al 13 Maggio 2012 il bilancio del 97% (98%) di tutte le entità (indirizzi) è minore di 10 BTC. I valori cambiano a 88% (91%) se si considera il valore massimale mai raggiunto durante l'intera vita dell'entità (indirizzo).

- 
- Solo 78 entità e 70 indirizzi hanno un bilancio finale maggiore di 10000 BTC. I numeri aumentano a 3812 entità e 3876 indirizzi se si osserva il bilancio massimale.
  - Il 97% (98%) delle entità (indirizzi) hanno meno di 10 transazioni a testa, mentre 75 entità e 80 indirizzi ne hanno almeno 5000.
  - La maggior parte delle transazioni sono piccole:
    - Il 28% delle transazioni nel grafo delle entità (47% in quello degli indirizzi) muove meno di 0.1 BTC.
    - Il 73% delle transazioni nel grafo delle entità (84% in quello degli indirizzi) muove meno di 10 BTC.
    - Esistono solo 364 transazioni nel grafo delle entità (340 in quello degli indirizzi) che muovono più di 50000 BTC.
  - Mt. Gox possiede il maggior numero di indirizzi (156722) ma non il maggior numero di transazioni (477526).
  - Una entità non meglio identificata che possiede il secondo maggior numero di indirizzi (il 50% di quelli di Mt. Gox, ovvero 246012) ha ricevuto il 31% in più di BTC rispetto a Mt. Gox (2886650 contro 2206170).
  - La pool Deepbit ha generato il 70% di transazioni in più rispetto a Mt. Gox (814044 in totale), pur possendo solo due indirizzi.

Un fatto interessante riguarda le transazioni con più di 50000 BTC. Esse sono esattamente 364, la prima delle quali datata 8 Novembre 2010 e valente 90000 BTC. Tracciando il flusso delle altre 363, i ricercatori si sono resi conto che 348 sono in realtà successori della prima grande transazione di Novembre. Il grafo che descrive tale transazione mostra catene di transazioni molto lunghe, suddivisione e riunificazioni di BTC e self-loop, tutti descritti nel dettaglio in [?] e molto probabilmente effettuati nel tentativo (evidentemente infruttuoso) di mascherare la destinazione e la provenienza del denaro.

## 7.2 Deflazione

Come si è detto, Bitcoin è un sistema monetario privo di inflazione, in quanto nessuna entità centrale può emettere moneta. Esiste però una elevatissima propensione alla deflazione<sup>1</sup> la quale potrebbe avere serie conseguenze per la sicurezza dell'intera rete. Dato il loro estremo valore in rapporto alle monete circolanti, le BTC tendono ad essere accumulate invece che spese sparendo quindi dal sistema dal punto di vista economico. Stessa fine fanno le monete per le quali è stata persa la chiave privata che permetteva loro di essere spese. Mano a mano che le BTC spariscono dalla

---

<sup>1</sup>Diminuzione generale del livello dei prezzi.

---

circolazione (per volontà o incidente), diminuisce il volume delle transazioni, quindi anche il numero di blocchi generati nell'unità di tempo e di conseguenza il numero di nuove monete introdotte in circolazione, il quale viene già dimezzato ad intervalli programmati in modo da generare in assoluto al massimo 21 milioni di BTC. L'attrattiva della creazione di nuovi blocchi diventa quindi via via sempre più ridotta, in quanto non solo ci saranno meno BTC generate, ma anche meno tasse da parte degli utenti a causa delle minori transazioni. Se la circolazione di BTC si riduce troppo, potrebbe verificarsi una perdita di interesse nel sistema, il che si traduce in un numero sempre minore di miners e quindi di "verificatori" per le transazioni, fino al punto in cui il sistema è diventato troppo debole per potersi difendere da fork nella blockchain o altri attacchi.

Purtroppo non esistono contromisure pratiche per il rischio di deflazione, e l'unico modo per rendere Bitcoin resistente a tale pericolo consiste nel rendere consapevoli gli utenti ed in particolare i minatori di quanto importante sia il loro contributo per la salvaguardia del loro stesso denaro.

Si può però imparare da Bitcoin per una futura moneta elettronica che sia resistente alla deflazione integrando un sistema di feedback per controllare il tasso di mining globale, in modo che rifornire il sistema nel caso la circolazione diminuisca.

## **7.3 Esempi reali ???titolo da cambiare**

### **7.3.1 Distributore automatico**

### **7.3.2 Bancomat**

### **7.3.3 Silkroad ??FORSE???**

## Appendice A

# Simulatore Bitcoin per la valutazione della privacy dell'utente

Il simulatore implementato da in [?] e utilizzato nella sezione 3.2.3 a pagina 40 è basato su turni: ad ogni turno (che corrisponde a circa una settimana simulata), alcuni eventi vengono aggiunti ad una lista di priorità con una probabilità determinata da un file di configurazione. Tali eventi corrispondono ad una delle seguenti operazioni:

**Nuova transazione** Un utente vuole effettuare una nuova transazione il cui orario, valore, scopo e beneficiario scaturisce dalle opzioni proposte dal file XML di configurazione. Il processo di gestione della transazione implementato nel simulatore è identico a quello presente nella rete Bitcoin.

**Generazione di un nuovo indirizzo** Gli utenti che sono stati configurati come sensibili alla propria privacy, periodicamente generano un nuovo indirizzo in cui trasferiscono parte delle loro monete per creare un po' di rumore nel pubLog. Tali indirizzi sono in aggiunta agli indirizzi ombra che vengono automaticamente creati per recuperare l'eventuale resto delle transazioni.

In conformità all'attuale situazione della rete, solo una minoranza di nodi sono anche *miners*, in quanto al giorno d'oggi tale attività è diventata redditizia solo se svolta su particolari hardware dedicati raggruppati nelle cosiddette *mining pools*. Si presume che tale hardware sia troppo oneroso o non necessario per la maggior parte dei frequentanti una università.

Il simulatore esclude alcune problematiche della rete Internet come le congestioni, i ritardi, il jitter, ecc. Si è assunto inoltre che tutte le transazioni generate sia ben formate e prive di tasse, e che non esistano tentativi di doppia spesa di BTC, sebbene transazioni mal formate e attacchi doppia spesa siano osservabili nella rete. Gli autori ritengono infatti che il comportamento malevolo di alcuni nodi nella rete non

---

sia rilevante per la loro indagine sulla privacy. Sebbene i blocchi vengono generati con una frequenza identica a quella della rete Bitcoin, per gli scopi del simulatore è stato raddoppiato l'intervallo di tempo medio tra i blocchi per meglio adattarsi alle dinamiche delle reti simulate, portandolo a circa 20 minuti tra un blocco e il successivo.



## Appendice B

# Probabilità di successo per l'attaccante

Il seguente sorgente rappresenta quello utilizzato da Nakamoto per calcolare la probabilità di successo di un attaccante in un attacco di double-spending (vedi ??). Il file `implement.h` è l'header utilizzato per ottenere i dati (risultati del tutto identici a quelli presentati dall'autore nel suo whitepaper) visualizzati nel grafico 4.1.

```
1 #include <math.h>
2 #include "implement.h"
3
4 double AttackerSuccessProbability(double q, int z)
5 {
6     double p = 1.0 - q;
7     double lambda = z * (q / p);
8     double sum = 1.0;
9     int i, k;
10    for (k = 0; k <= z; k++)
11    {
12        double poisson = exp(-lambda);
13        for (i = 1; i <= k; i++)
14            poisson *= lambda / i;
15        sum -= poisson * (1 - pow(q / p, z - k));
16    }
17    return sum;
18 }
```



# Bibliografia

- [1] *Bitcoin Script*. <https://en.bitcoin.it/wiki/Script>.
- [2] *Bitcoin Wiki*. <https://en.bitcoin.it/wiki/>.
- [3] *BlockChain.info*. <https://www.blockchain.info>.
- [4] *Mt. GOX*. <https://mtgox.org>.
- [5] Abdalla, Michel, Xavier Boyen, Céline Chevalier e David Pointcheval: *Distributed Public-Key Cryptography from Weak Secrets*. Nel Jarecki, Stanislaw e Gene Tsudik (curatori): *Public Key Cryptography PKC 2009*, volume 5443 della serie *Lecture Notes in Computer Science*, pagine 139–159. Springer Berlin Heidelberg, 2009, ISBN 978-3-642-00467-4. [http://dx.doi.org/10.1007/978-3-642-00468-1\\_9](http://dx.doi.org/10.1007/978-3-642-00468-1_9).
- [6] Androulaki, Elli, Ghassam Karame, Marc Roeschlin, Tobias Scherer e Srdjan Capkun: *Evaluating User Privacy in Bitcoin*. Cryptology ePrint Archive, Report 2012/596, 2012. <http://eprint.iacr.org/2012/596.pdf>.
- [7] Babaioff, Moshe, Shahar Dobzinski, Sigal Oren e Aviv Zohar: *On bitcoin and red balloons*. Nel *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, pagine 56–73, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1415-2. <http://doi.acm.org/10.1145/2229012.2229022>, Disponibile in download gratuito all'indirizzo <http://arxiv.org/pdf/1111.2626.pdf>.
- [8] Back, Adam: *Hashcash - a denial of service counter-measure*, 2002. <http://www.hashcash.org/papers/hashcash.pdf>.
- [9] Bamert, Tobias, Christian Decker, Lennart Elsen, Roger Wattenhofer e Samuel Welten: *Have a Snack, Pay with Bitcoins*. Nel *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Trento, Italy, September 2013. [http://www.tik.ee.ethz.ch/file/848064fa2e80f88a57aef43d7d5956c6/P2P2013\\_093.pdf](http://www.tik.ee.ethz.ch/file/848064fa2e80f88a57aef43d7d5956c6/P2P2013_093.pdf).
- [10] Barber, Simon, Xavier Boyen, Elaine Shi e Ersin Uzun: *Bitter to Better - How to Make Bitcoin a Better Currency*. Nel Keromytis, Angelos D. (curatore):

- 
- Financial Cryptography and Data Security*, volume 7397 della serie *Lecture Notes in Computer Science*, pagine 399–414. Springer Berlin Heidelberg, 2012, ISBN 978-3-642-32945-6. [http://dx.doi.org/10.1007/978-3-642-32946-3\\_29](http://dx.doi.org/10.1007/978-3-642-32946-3_29), Disponibile in download gratuito all'indirizzo <http://www.cs.stanford.edu/~xb/fc12/>.
- [11] Boyen, Xavier: *Halting password puzzles*. Nel *Proc. Usenix Security*, 2007. [http://static.usenix.org/events/sec07/tech/full\\_papers/boyen/boyen.pdf](http://static.usenix.org/events/sec07/tech/full_papers/boyen/boyen.pdf).
- [12] Clark, Jeremy e Aleksander Essex: *CommitCoin: Carbon Dating Commitments with Bitcoin*. Nel Keromytis, Angelos D. (curatore): *Financial Cryptography and Data Security*, volume 7397 della serie *Lecture Notes in Computer Science*, pagine 390–398. Springer Berlin Heidelberg, 2012, ISBN 978-3-642-32945-6. [http://dx.doi.org/10.1007/978-3-642-32946-3\\_28](http://dx.doi.org/10.1007/978-3-642-32946-3_28), Disponibile in download gratuito all'indirizzo <http://eprint.iacr.org/2011/677.pdf>.
- [13] Decker, Christian e Roger Wattenhofer: *Information Propagation in the Bitcoin Network*. Nel *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Trento, Italy, September 2013. [http://www.tik.ee.ethz.ch/file/49318d3f56c1d525aabf7fda78b23fc0/P2P2013\\_041.pdf](http://www.tik.ee.ethz.ch/file/49318d3f56c1d525aabf7fda78b23fc0/P2P2013_041.pdf).
- [14] Engle, Marling e Javed I. Khan: *Vulnerabilities of P2P Systems and a Critical Look at their Solution*. Rapporto Tecnico, Kent State University, Networking and Media Communications Research Laboratories, Department of Computer Science, 233 MSB, Kent, OH 44242, November 2006. <http://medianet.kent.edu/technicalreports.html>.
- [15] Hearn, Mike: *Bitcoin Contracts*. <https://en.bitcoin.it/wiki/Contracts>.
- [16] Karame, Ghassan, Elli Androulaki e Srdjan Capkun: *Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin*. IACR Cryptology ePrint Archive, 2012:248, 2012. <http://people.scs.carleton.ca/~clark/biblio/bitcoin/Karame%202012.pdf>.
- [17] Kshemkalyani, Ajay D. e Mukesh Singhal: *Peer-to-peer computing and overlay graphs*, capitolo 18. Cambridge University Press, The Edinburgh Building, Cambridge CB2 8RU, UK, first edizione, 2008.
- [18] Kurose, James F. e Keith W. Ross: *Applicazioni peer-to-peer*, capitolo 2, pagine 131–144. Pearson Education, Addison-Wesley, quarta edizione, 2006.
- [19] Merkle, Ralph C.: *Protocols for Public Key Cryptosystems*. Nel *IEEE Symposium on Security and privacy*, volume 1109, pagine 122–134, April 1980. <http://www.cs.washington.edu/research/projects/poirot3/Oakland/sp/PAPERS/00044729.PDF>.

- 
- [20] Moore, Tyler e Nicola Christian: *Beware the Middleman: Empirical Analysis of Bitcoin-Exchange Risk*. Nel *Proceedings of Financial Cryptography 2013*, April 2013. <http://www.truststc.org/pubs/907.html>, Disponibile in download gratuito all'indirizzo <http://fc13.ifca.ai/proc/1-2.pdf>.
- [21] Nakamoto, Satoshi: *Bitcoin: A Peer-to-Peer Electronic Cash System*. [satoshin@gmx.com](mailto:satoshin@gmx.com). [www.bitcoin.org](http://www.bitcoin.org).
- [22] Nielsen, Michael: *How Bitcoin actually work*. <http://www.michaelnielsen.org/ddi/how-the-bitcoin-protocol-actually-works/>.
- [23] Reid, Fergal e Martin Harrigan: *An Analysis of Anonymity in the Bitcoin System*. Nel Altshuler, Yaniv, Yuval Elovici, Armin B. Cremers, Nadav Aharony e Alex Pentland (curatori): *Security and Privacy in Social Networks*, pagine 197–223. Springer New York, 2013, ISBN 978-1-4614-4138-0. [http://dx.doi.org/10.1007/978-1-4614-4139-7\\_10](http://dx.doi.org/10.1007/978-1-4614-4139-7_10), Disponibile in download gratuito all'indirizzo <http://arxiv.org/pdf/1107.4524.pdf>.
- [24] Ron, Dorit e Adi Shamir: *Quantitative Analysis of the Full Bitcoin Transaction Graph*. Cryptology ePrint Archive, Report 2012/584, 2012. <http://eprint.iacr.org/2012/584.pdf>.
- [25] Rosenfeld, Meni: *Analysis of Bitcoin Pooled Mining Reward Systems*. CoRR, abs/1112.4980, 2011. Disponibile in download gratuito all'indirizzo <http://arxiv.org/pdf/1112.4980v1.pdf>.
- [26] Rosenfeld, Meni: *Analysis of hashrate-based double-spending*. 2012. <https://bitcoil.co.il/Doublespend.pdf>.
- [27] Schoeder, Detlef, Kai Fischbach e Christian Schmitt: *Core Concepts in Peer-to-Peer Networking*, capitolo 1. Idead Group Inc., 2005.
- [28] Szabó, Nick: *Formalizing and Securing Relationships on Public Networks*. <http://szabo.best.vwh.net/formalize.html>.