UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Bitcoin e Blockchain

Esame di Crittografia

A cura di Gianluca Scatigna

[Matricola 718633]

ANNO ACCADEMICO 2019-2020

Sommario

1. Introduzione 3
2. Blockchain 4
Stutture dati con hash pointers 5
Merkle tree 5
3. Proof Of Work 7
Hashcash 7
4. Mining 9
Validazione di un blocco 10
Collegamento e Blockchain Fork 10
Consenso Distribuito 11
5. Bibliografia 12

1. Introduzione

Bitcoin è la prima valuta digitale decentralizzata che non è sostenuta da alcun governo o organizzazione. Nonostante la mancanza di una organizzazione centrale che la presiede è in grado di consentire transazioni istantanee peer-to-peer, non ha bisogno di una terza parte fiduciaria per operare, basa la sicurezza su protocolli crittografici e quindi rappresenta low-cost banking per tutti, ovunque. Bitcoin ha avuto i suoi precursori, a partire dal 1982 con Ecash, ma nessun progetto ha mai avuto successo.
L'annuncio da parte di uno sconosciuto, Satoshi Nakamoto, avvenne il 31 ottobre 2008 scrivendo a una mailing list di crittografia e pubblicando "Bitcoin: A Peer-to-Peer Electronic Cash System". Nel documento sosteneva di aver lavorato e creato Bitcoin ma soprattutto di aver risolto il problema della doppia spesa, definito fino ad allora irrisolvibile[2].
Il 3 gennaio 2009 fu minato il blocco genesi dando vita a Bitcoin e poco dopo venne rilasciato il suo codice sorgente. Bitcoin è stato il primo progetto con successo ad implementare la tecnologia della blockchain: un libro mastro in cui sono registrate tutte le transazioni Bitcoin.

Con riferimento alla blockchain di Bitcoin, al momento l'unica tra le tante realmente valida dato che il suo protocollo non è mai stato penetrato, questo documento esamina nel dettaglio la sua struttura dati e ciò che la compone ovvero la struttura del blocco e la motivazione della scelta del Merkle tree come struttura dati per aggregare le transazioni Bitcoin.
Due capitoli sono dedicati all'analisi del proof-of-work, elencando le proprietà delle funzioni di costo e l'algoritmo Hashcash anticipato da alcuni cenni storici sulla sua origine. Con il fine di dimostrare la relazione tra i parametri difficulty e hash target è stata eseguita una estrazione di dati dalla blockchain.
Nell'ultimo capitolo vi è l'analisi dei processi che consentono alla Bitcoin network di godere del consenso globale pur essendo decentralizzato. Il primo processo analizzato è il mining con l'applicazione dell'algoritmo Hashcash, successivamente le condizioni per la validazione di un blocco di transazioni e la gestione dei fork temporanei. L'ultimo paragrafo definisce il consenso distribuito ed una sua vulnerabilità definito "attacco del 51%".

2. Blockchain

La blockchain è una struttura dati realizzata con una **lista concatenata** dove ogni blocco ha un puntatore hash al blocco precedente, detto blocco padre. Visualizzando la blockchain come uno stack allora il primo blocco, detto blocco genesi, forma la base e i successivi blocchi si sovrappongono ad esso aumentando l'altezza della catena. Ogni blocco può essere identificato in maniera univoca dal **block header hash** calcolato eseguendo un **DOUBLE-SHA-256** del block header, ciò rappresenta il risultato dell'algoritmo di Proof Of Work eseguito dai nodi minatori della rete.

Il blocco è una struttura dati contenitore formata da una testa, **block header**, contenente metadati, seguita da una lista di transazioni. Il block header ha una dimensione massima di 80 bytes, una singola transazione è di almeno 250 bytes e dato che un blocco contiene in media più di 500 transazioni ne risulta che queste ultime determinano la dimensione totale dell'intero blocco.

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header
1–9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

Figura 1 Struttura del blocco

Il block header è formato da tre set di metadati:

1. Il puntatore hash al blocco genitore
2. La difficoltà, il timestamp e il nonce
3. La radice del merkle tree

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The Proof-Of-Work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the Proof-Of-Work algorithm

Figura 2 Struttura del block header

Stutture dati con hash pointers

Un **Hash pointer** può essere usato in qualsiasi struttura dati basata su puntatori senza cicli. Utilizzati su una lista concatenata si ottiene la blockchain mentre su un albero binario si ottiene il **Merkle tree**. Il puntatore Hash oltre a referenziare il blocco o nodo padre ne garantisce l'integrità e l'immutabilità dell'intera struttura dati. L'alterazione dei dati di un blocco padre, in una blockchain, implica un cambiamento del suo block header hash che a sua volta impone un aggiornamento del puntatore hash nel blocco figlio e di conseguenza anche del suo block header hash. L'effetto a cascata si ripercuote sino all'ultimo blocco e ne consegue che è impossibile alterare retroattivamente un singolo blocco lasciando immutati i successivi.

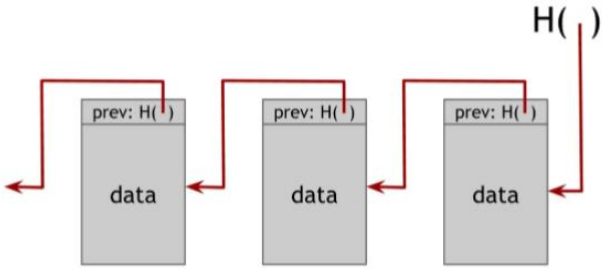

Figura 3 Struttura dati della Blockchain

Merkle tree

Il Merkle tree o **albero binario di hash** è una struttura dati usata per raggruppare efficientemente e verificare l'integrità di grandi set di dati. L'albero è costruito calcolando ricorsivamente l'hash della concatenazione di coppie di nodi sino ad ottenere un unico hash ossia la radice. Per bilanciare un albero con un numero di foglie dispari l'ultimo nodo viene duplicato e accoppiato con se stesso.

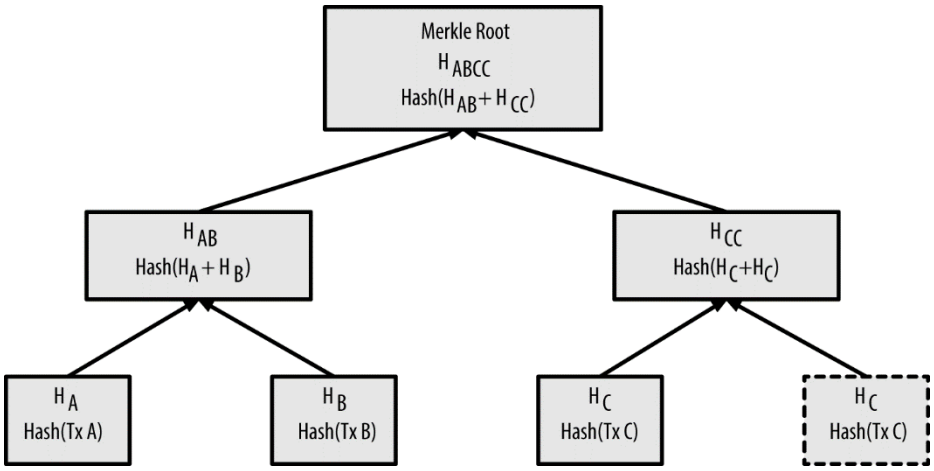

Figura 4 Struttura dati del Merkle tree

Ad esempio, in figura 4 il nodo padre H_{AB} è calcolato applicando un DOUBLE-SHA-256 ai due nodi figli ovvero alle transazioni H_A e H_B. Successivamente H_A è concatenato ad H_B formando una stringa di 64 bytes cui verrà applicato il DOUBLE-SHA-256.

- H_A = SHA256(SHA256(Transaction A))
- H_B = SHA256(SHA256(Transaction B)
- H_{AB} = SHA256(SHA256(H_A + H_B))

Il Merkle tree garantisce l'inclusione di una transazione in un blocco producendo un **authentication path o merkle path** che connette la specifica transazione alla radice dell'albero. Il merkle path è calcolato da un full node producendo **log2(N) hash**, dove N è il numero di foglie dell'albero.

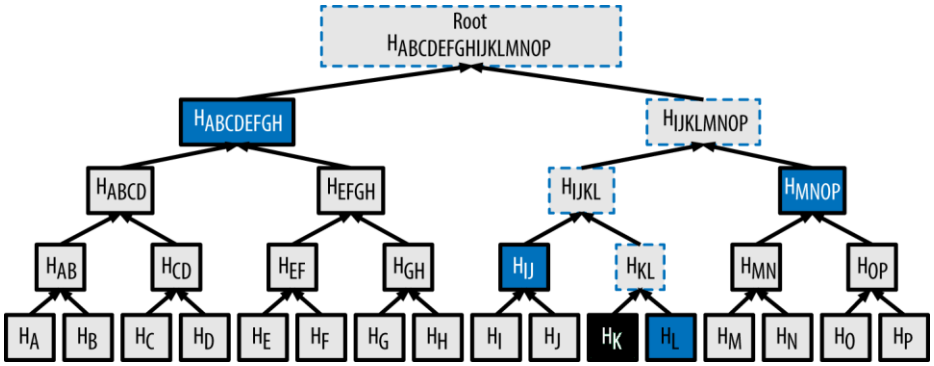

Figura 5 Nodi di un Merkle path

I meccanismi forniti dal Merkle tree permettono ad alcuni nodi della rete Bitcoin, detti **SPV**, di memorizzare una versione più leggera della blockchain formata solo dai block header.
Un nodo SPV per dimostrare che l'hash della transazione K appartiene al blocco con 16 transazioni, figura 5, avrà bisogno di un merkle path con quattro hash ovvero dei nodi H_L, H_{IJ}, H_{MNOP} e $H_{ABCDEFGH}$.
Il nodo SPV formalizza e inoltra la richiesta del merkle path, in maniera anonimizzata utilizzando i filtri bloom, ai full nodes i quali memorizzano l'intera blockchain. Il nodo SPV alla ricezione del merkle path può ricalcolare autonomamente il ramo dell'albero dalla transazione interessata fino alla merkle root e di conseguenza verificare che questa appartenga al blocco e quest'ultimo appartenga alla blockchain.
La funzione logaritmo in base 2 tende a crescere molto lentamente consentendo di produrre in maniera efficiente una path di 16 hash con dimensione di 512 bytes per un blocco di 16 megabytes con 65,535 transazioni.

Number of transactions	Approx. size of block	Path size (hashes)	Path size (bytes)
16 transactions	4 kilobytes	4 hashes	128 bytes
512 transactions	128 kilobytes	9 hashes	288 bytes
2048 transactions	512 kilobytes	11 hashes	352 bytes
65,535 transactions	16 megabytes	16 hashes	512 bytes

Figura 6 Efficienza del Merkle tree

3. Proof Of Work

Un sistema Proof Of Work (**POW**) impone al richiedente di un servizio l'esecuzione di una funzione di costo della CPU, detta anche puzzle computazionale, la quale è computazionalmente costosa ma verificabile in maniera efficiente.
Nel contesto di una funzione di costo:

- Il client computa il token eseguendo la funzione di costo **MINT()**
- Il server riceve il token e ne calcola il valore utilizzando la funzione **VALUE()**
- Le funzioni sono parametrizzate dal carico di lavoro che il client spende in media per computare un token

Proprietà di una funzione di costo[3]:

1. Interattività
   Una funzione di costo è interattiva se il server invita il client allo svolgimento di una **challenge C**. Il server usa la funzione **CHAL()** per computare la challenge.
   Una funzione di costo è non interattiva se il client sceglie la propria challenge o se utilizza valori randomici per la funzione MINT().
2. Verificabilità
   Una funzione di costo è **verificabile pubblicamente** se può essere verificata efficientemente da terze parte senza avere accesso a informazioni segrete. L'efficienza implica che il consumo di risorse utilizzate nell'esecuzione della funzione VALUE() è inferiore a quello della funzione MINT().
3. Costo fisso o probabilistico
   Una funzione di costo è a **costo fisso** se utilizza un carico di risorse predeterminato per la computazione.
   Una funzione di costo con **costo probabilistico** ha un tempo di calcolo atteso predicibile. Il costo probabilistico è di tipo limitato se lo spazio di ricerca di una soluzione è finito viceversa è di tipo illimitato.
4. Trapdoor-Free
   Una funzione di costo è definita **trapdoor-free** se il challenger non ha vantaggi dalla computazione dei token poiché una funzione nota potrebbe essere usata dal challenger per creare tokens di valore arbitrario spendendo meno risorse del necessario.

Hashcash

Nel maggio del 1997 fu proposto da **Adam Back** un meccanismo per ridurre l'abuso delle risorse internet commesso dai mail spammers. Per risolvere tale problema l'autore, ignaro di una ricerca simile condotta nel 1992 da **Cynthia Dwork, Andrew Goldberg e Moni Naor**[4], propose una funzione di costo, chiamata **Hashcash**, in grado di computare un token utilizzabile come Proof-Of-Work.
Hashcash è una funzione di costo non interattiva, verificabile pubblicamente, trapdoor-free e con costo probabilistico illimitato[3]. La versione 0 del protocollo Hashcash (1997) si basa sulla ricerca della seconda preimmagine parziale utilizzando SHA-1. Successivamente, nella versione 1 del 2002, **Hal Finney e Thomas Boschloo** proposero di ricercare la collisione rispetto ad una stringa di output fissata e i cui primi k bits siano 0 usando SHA-256[5]. Il vantaggio della versione 1 rispetto alla

precedente è l'ottimizzazione del processo di verifica del Proof-Of-Work poiché è richiesto il calcolo di un solo hash mentre invece il calcolo avviene due volte nella versione originale[3].
L'algoritmo Proof-Of-Work di Bitcoin utilizza la versione 1 di Hashcash la cui stringa target, oggetto della collisione, è determinata da un parametro variabile detto **difficulty**.
Il parametro difficulty ha lo scopo di garantire che la generazione di un blocco di transazioni della blockchain avvenga in un intervallo di tempo di circa 10 minuti. Tecnicamente ciò si traduce in un aumento della complessità di computazione se la generazione del blocco avviene in un tempo inferiore ai 10 minuti, viceversa in un decremento. **Satoshi Nakamoto**, padre di Bitcoin, ipotizzò che tale arco di tempo fosse un compromesso per garantire robustezza alla blockchain e performance per il bitcoin network. In sintesi, si può affermare che tale parametro bilancia la crescita esponenziale dell'hashing power della rete bitcoin.
Tutti i nodi della rete ogni 2016 blocchi, circa due settimane, ricalcolano il parametro difficulty eseguendo una semplice equazione:

Nuovo target = Vecchio Target * (Intervallo di tempo degli ultimi 2016 blocchi / 20160 minuti)

Il nuovo target, per evitare una estrema volatilità del parametro difficulty, dovrà essere inferiore al massimo di un quarto rispetto al precedente.

Date	Difficulty	Hash Target
06/07/2010 00:00	24	000000000ae492000000000000000000000000000000000000000000000000000
27/03/2011 00:00	68.978	000000000000f338000000000000000000000000000000000000000000000000000
05/05/2011 00:00	109.670	00000000000098fa000000000000000000000000000000000000000000000000000
25/06/2011 00:00	1.379.192	00000000000000c2a11000000000000000000000000000000000000000000000000
05/09/2012 00:00	2.440.643	00000000000006dfbe000000000000000000000000000000000000000000000000
17/02/2013 00:00	3.275.465	000000000000051f3b000000000000000000000000000000000000000000000000
31/03/2013 00:00	6.695.826	0000000000002816d000000000000000000000000000000000000000000000000
27/05/2013 00:00	12.153.412	0000000000001616400000000000000000000000000000000000000000000000
22/08/2013 00:00	50.810.339	000000000000005487320000000000000000000000000000000000000000000000
15/09/2013 00:00	112.628.549	000000000000026222200000000000000000000000000000000000000000000000
26/12/2013 00:00	1.180.923.195	000000000000003a30c00000000000000000000000000000000000000000000000
15/06/2014 00:00	11.756.551.917	000000000000000005d859900000000000000000000000000000000000000000000
24/11/2014 00:00	40.300.030.328	000000000000000001b486100000000000000000000000000000000000000000000
13/11/2015 00:00	65.848.255.180	000000000000000010b289000000000000000000000000000000000000000000000
30/01/2016 00:00	120.033.340.651	000000000000000000928ef00000000000000000000000000000000000000000000
13/06/2016 00:00	196.061.423.939	000000000000000059ba00000000000000000000000000000000000000000000000
06/05/2017 00:00	521.974.519.553	000000000000000021b3e00000000000000000000000000000000000000000000000
06/09/2017 00:00	903.315.386.419	000000000000000013798000000000000000000000000000000000000000000000000
25/01/2018 00:00	2.460.585.023.617	000000000000000000072644b00000000000000000000000000000000000000000000
27/09/2018 00:00	7.152.633.351.906	000000000000000000275a1f00000000000000000000000000000000000000000000
03/12/2019 00:00	12.973.235.968.800	000000000000000000015b23d00000000000000000000000000000000000000000000
29/01/2020 00:00	15.466.098.935.555	000000000000000000001232fe00000000000000000000000000000000000000000000

Figura 7 Relazione tra i parametri Difficulty e Hash Target

I dati della figura 7 sono stati estratti dalla blockchain di bitcoin e dimostrano la relazione tra il parametro difficulty e l'hash target, oggetto della collisione parziale. Dalla tabella si evince che ad una maggiore difficoltà corrisponde un hash target che si avvicina allo zero. Si potrebbe affermare, in maniera semplicistica, che maggiore è il numero di zeri nelle cifre più significative della stringa hash target e maggiore sarà il valore difficulty e dunque la complessità computazionale richiesta. Utilizzando Hashcash, un blocco candidato di transazioni Bitcoin è considerato valido dai nodi minatori se l'hash del suo block header è inferiore all'hash target.

4. Mining

Il seguente paragrafo fa riferimento all'attività di mining nel sistema di Bitcoin.

L'attività di mining garantisce la decentralizzazione della blockchain in una rete peer-to-peer formata da nodi anonimi ed è incentivata da una ricompensa, in origine 50 bitcoin, per la validazione di un singolo blocco. La ricompensa non è immutabile nel tempo ma anzi si dimezza ogni 4 anni per effetto di un processo chiamato halving il quale è alla base del modello economico di Bitcoin.

I nodi minatori validano in maniera indipendente gli uni dagli altri le nuove transazioni verificandole su una lunga lista di condizioni, dettagliate nelle funzioni **AcceptToMemoryPool**, **CheckTransaction** e **CheckInputs** in **Bitcoin Core**. Ogni transazione dopo essere stata validata è aggregata assieme ad altre transazioni non confermate in un blocco, detto **memory pool** o **transaction pool**, in attesa di essere

minato. Il mining di un blocco implica l'applicazione dell'algoritmo Hashcash sul block header e l'hash del block header dipende dai tre set di metadati che lo compongono, figura 2.
Il nodo minatore al fine di trovare la collisione esegue l'hash del blocco cambiando di volta in volta il valore del parametro nonce. Questo ha una dimensione di **4 bytes** definendo uno **spazio di soluzioni di 2^{32}** valori di hash possibili. A partire dal 2012 i soli 4 bytes del nonce non risultarono più sufficienti per minare un blocco poiché a causa dell'elevata difficoltà raggiunta lo spazio non contiene soluzioni. Per risolvere tale problema è stato necessario trovare una nuova fonte di entropia. Una transazione particolare presente in ogni blocco poiché contiene la ricompensa per il minatore, detta coinbase transaction, risulta adatta come **extra nonce** perché ha a disposizione **8 bytes** di spazio dati da usare in maniera arbitraria. Lo spazio di soluzioni di hash possibili diventa dunque di **12 bytes** ossia **2^{96}**.
È bene precisare che i nodi minatori non sono alla ricerca della stessa soluzione ossia della stessa stringa per una collisione parziale. È molto probabile che ogni nodo abbia aggregato transazioni diverse o in diverso ordine cambiando il valore della radice del merkle tree. In ogni caso l'unicità del block header hash è data dalla coinbase transaction poiché questa contiene l'indirizzo bitcoin dove il minatore riceverà la ricompensa[6].

Validazione di un blocco

Ogni nodo della rete valida, **in maniera indipendente**, un blocco ricevuto verificando una lista di condizioni tra cui se:

- La struttura del blocco è sintatticamente valida
- Il block header hash è inferiore al target
- Il timestamp è inferiore di due ore rispetto all'orario corrente
- La dimensione del blocco non eccede i limiti
- Solo la prima transazione è una coinbase transaction
- Tutte le transazioni del blocco risultano valide, come discusso nel capitolo precedente

La validazione indipendente di ogni blocco da parte di ogni nodo della rete assicura che i nodi minatori non possano barare. Ad esempio, se un nodo minatore malevolo alterasse la coinbase transaction del proprio blocco, assegnandosi una ricompensa più alta, nessun altro nodo minatore considererà tale blocco valido e verrebbe rifiutato. Il nodo malevolo avrà così sprecato la propria computazione senza ottenere alcun guadagno.

Collegamento e Blockchain Fork

Un nodo dopo aver ottenuto la validazione del blocco proverà a collegarlo con la testa della catena ossia con l'ultimo blocco già considerato valido. Nel caso ci fossero più catene di blocchi, il nodo considera come catena principale quella con maggior Proof-Of-Work, solitamente la catena più lunga. La decentralizzazione della blockchain, tuttavia, produce inconsistenze tra le copie locali dei nodi a causa della ricezione dei blocchi in orari differenti. L'effetto dell'inconsistenza è un **fork temporaneo** della catena che solitamente si risolve con il mining del blocco successivo. Un fork temporaneo, in particolare, avviene nel momento in cui almeno due nodi minatori validano blocchi diversi in un breve arco di tempo. Entrambi i blocchi, effettivamente validi, vengono propagati dai nodi formando due blockchain differenti. Il fork sarà risolto dal nodo che validerà il blocco successivo facendo convergere l'intera rete su una singola blockchain poiché quest'ultima ha il Proof-Of-Work più elevato, infine, le transazioni del **blocco orfano**, quello scartato, saranno reinserite nella **mempool** per essere incluse nel blocco successivo.

Consenso Distribuito

La rete di Bitcoin ottiene il consenso globale senza una autorità centrale.

I nodi della rete, come visto nei paragrafi precedenti, operano sempre in maniera indipendente seguendo un insieme di regole comuni a tutti.
Il consenso decentralizzato è ottenuto con i processi prima analizzati:

1. Verifica indipendente delle transazioni
2. Aggregazione indipendente delle transazioni in un blocco
3. Proof-Of-Work individuale
4. Validazione indipendente del blocco
5. Selezione indipendente della catena con maggior Proof-Of-Work

Il consenso distribuito inoltre **disincentiva qualsiasi comportamento malevolo** se la maggioranza dei nodi minatori si comporta onestamente. Infatti, un noto scenario teorico di attacco al meccanismo del consenso è detto "**attacco del 51%**" dove un gruppo di minatori controlla la maggioranza dell'hashing power della rete. Il gruppo malevolo, in questo caso, potrebbe alterare le transazioni a proprio vantaggio o causare denial of service. Tuttavia, la mole di risorse richiesta risulta così onerosa tale da non essere profittevole. Nonostante la sua nominazione, un attacco di questo tipo ha una garanzia di successo totale se la maggioranza dei nodi malevoli detiene il controllo del 51% dell'hashing power ma non risulta essere un requisito indispensabile poiché, in teoria, sarebbe sufficiente anche il 30% per realizzare alcuni disservizi. In sintesi, l'attacco ha maggior garanzia di successo tanto più è alto il controllo della rete da parte dei nodi malevoli.

5. Bibliografia

[1] Antonopoulos A. M., Mastering Bitcoin - Programming the Open Blockchain 2E 2017.

[2] Satoshi Nakamoto, “A Peer-to-Peer Electronic Cash System“, bitcoin.org/bitcoin.pdf.

[3] Adam Back, Hashcash - A Denial of Service Counter-Measure, 2002.

[4] Cynthia Dwork, Andrew Goldberg e Moni Naor, On Memory-Bound Functions for Fighting Spam, 1992.

[5] Hal Finney, Thomas Boschloo, Personal communication, 2002.

[6] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder, Bitcoin and Cryptocurrency Technologies, 2016.