Matematica, codice e crittografia in Bitcoin

Luca Polverini
email luca.polverini@gmail.com

2020

Indice

Parte I

Bitcoin

Capitolo 1

Teoria

1.1 Introduzione

Bitcoin è una valuta digitale *sperimentale* che consente pagamenti istantanei senza confini (*border-less*) verso "chiunque" nel mondo e senza ricorso ad una autorità centrale (la banca).

Nel corso della storia moderna sono stati effettuati vari tentativi di creare una valuta digitale, per lo più fallimentari.

Nel 2008, Satoshi Nakamoto [1] pubblica un *white-paper*[2] mettendo in luce un metodo per risolvere il problema della doppia spesa (*double-spending*) in un contesto decentralizzato, senza ricorrere alla "zecca", dando così vita alla tecnologia *block-chain* in generale e alle cripto-valute in particolare.

Le caratteristiche fondanti della tecnologia blockchain sono essenzialmente tre:

- architettura distribuita *peer-to-peer*
- ricorso alla *crittografia* (hashing e firma digitale)
- risoluzione del *Problema del Generale Bizantino* (*double-spending*) attraverso meccanismi di consenso (*Proof of Work* in Bitcoin)

1.1.1 Architettura punto a punto

L'architettura punto-a-punto consente di distribuire il libro mastro (*ledger*) delle transazioni su tutti i nodi della rete, indipendentemente da una organizzazione centrale. Per fare un parallelo, quando visitiamo un sito web richiediamo e riceviamo contenuti da "un server centrale" (ad esempio Facebook o Google); nel caso punto-a-punto non esiste un server centrale ma una collezione di nodi indipendenti che replicano i contenuti e che possono fornirli su richiesta. Ne consegue che i contenuti non sono gestiti da un unico soggetto ma sono custoditi, gestiti e resi disponibili da molti soggetti fra loro indipendenti.

[1] (uno pseudonimo dietro al quale si cela un lui, un lei, un gruppo di persone o una associazione)

1.1.2 Crittografia

Hash

La funzione di *hash* consente di associare dati di arbitraria lunghezza a codici univoci di lunghezza definita (e.g. 256 bit). Possiamo pensare all'hash come una mappa unidirezionale che a partire da un contenuto ne deriva in maniera deterministica un indice univoco di lunghezza finita. Il contenuto prende il nome di pre-immagine e l'hash prende il nome di immagine. Ad ogni pre-immagine corrisponde una ed una sola immagine (ai fini pratici). L'algoritmo di hasing è veloce ma non è computazionalmente possibile la ricostruzione della pre-immagine a partire dal codice hash e la probabilità che due contenuti diano luogo allo stesso hash, utilizzando algoritmi sicuri (tipo shasum-256) è praticamente irrilevante. Inoltre, piccole variazioni del contenuto danno luogo a hash molto differenti. Questo permette di referenziare utenti, transazioni e blocchi per mezzo di codici crittograficamente sicuri.

Ricapitolando, dato un messaggio, idealmente il suo hash:

- è univoco e deterministico dato un messaggio esiste uno ed un solo hash (in pratica)
- è unidirezionale, altamente improbabile risalire al messaggio originale
- è veloce da computare

Firma digitale

La firma digitale (correntemente ECDSA in Bitcoin) consente di:

- dimostrare la proprietà di moneta (UTXO) attraverso la coppia chiave pubblica e privata
- firmare il trasferimento di moneta verso un secondo soggetto attraverso l'impiego della chiave privata
- consentire ai nodi partecipanti della rete la verifica della liceità delle operazioni

1.1.3 Consenso

Il meccanismo di consenso permette ai nodi di concordare l'estensione del libro mastro inserendo nuove transazioni nella block-chain.

Le transazioni **valide** inviate dagli utenti vengono propagate dai nodi a tutta la rete. I nodi di tipo *miner* (minatori), come vedremo nel seguito, raccolgono le transazioni e competono per costruire un nuovo blocco che le contenga da aggiungere alla catena.

Ne deriva che ogni neo-blocco è una entità che contiene le transazioni che vi sono state inserite dal nodo miner che è riuscito per primo a crearlo. Per essere aggiunto alla catena il blocco deve possedere i requisiti di validità previsti dal protocollo e deve essere validato da tutti i nodi della rete attraverso un **meccanismo di consenso**.

Il meccanismo di consenso di Bitcoin prende il nome di *Proof of Work*. Per brevità, nel seguito, ci riferiremo alla *Proof of Work* con l'acronimo **PoW**.

L'algoritmo PoW è progettato in modo tale da rendere altamente improbabile la modifica fraudolenta della catena se non possedendo almeno il 51% delle risorse hardware dell'intera rete (una cpu, un voto [2])[2];

[2] in ottica evolutiva, nel seguito, esamineremo alcune non linearità

PoW è difficile, non complicato; si tratta di un algoritmo semplice (computazione di codici hash) ma ad ogni tentativo di "soluzione" corrisponde una probabilità di successo molto bassa (modulata programmaticamente dal protocollo per produrre circa 2016 blocchi ogni due settimane) risultando di fatto molto dispendioso in termini di tempo, risorse hardware ed energetiche e fa parte degli stratagemmi progettuali atti a garantire la sicurezza della catena in un contesto distribuito.

Se da un lato la "difficoltà" della PoW serve per proteggere l'integrità della catena (libro mastro o *ledger*) richiedendo risorse hardware consistenti, dall'altro consente ai nodi minatori di essere remunerati per la loro attività. La creazione di un nuovo blocco remunera il nodo miner attraverso un sussidio e le commissioni associate alle transazioni.

Si può affermare che Bitcoin faccia sapiente uso della teoria dei giochi [3] per garantire sicurezza ed equilibrio nel quadro di un insieme di nodi privi di fiducia reciproca, ove l'interesse individuale converge verso quello collettivo date le regole scritte nel codice stesso. Il meccanismo di consenso è una di queste regole.

1.1.4 Contenuti del testo

In questo testo analizzeremo Bitcoin, il codice di bitcoin-core ed esploreremo dei metodi per simulare l'attività di mining. In particolare, il capitolo () mostra un metodo per la computazione della remunerazione attesa dell'attività di mining di criptovalute.

[3] sempre in ottica evolutiva, esistono critiche circa le tendenze centralizzatrici

1.2 La funzione di Hash

La funzione di hash è una funzione non invertibile che mappa dati di lunghezza arbitraria in valori di lunghezza predefinita. Il dato sui quali si applica la funzione prende il nome di pre-immagine, il risultato della funzione, detto hash (o valore di hash, codice di hash, digest) ne è l'immagine. L'immagine (hash) è un valore che in rappresentazione binaria è dato da numero definito di bit (ad esempio, a seconda dell'algoritmo utilizzato, 128, 160, 256, etc ...).

In crittografia la funzione di hash deve soddisfare, fra le altre, le seguenti proprietà:

- determinismo fissando la pre-immagine la funzione di hash genera sempre la stessa immagine,
- efficienza: deve essere veloce da computare,
- resistenza alla pre-immagine, cioè sia computazionalmente intrattabile la ricerca di una stringa che dia come risultato l'immagine.
- resistenza alla seconda pre-immagine, ovvero che sia computazionalmente intrattabile la ricerca di una stringa il cui hash sia uguale a quello di una stringa data
- resistenza alle collisioni, è richiesto che sia computazionalmente intrattabile la ricerca di due stringhe fra loro diverse che diano luogo alla stessa immagine.
- effetto valanga: una piccola variazione della pre-immagine produce valori di hash molto diversi che appaiono incorrelati.

Esempi SHASUM256 e RIPEMD160 sono due funzioni di hash utilizzate in Bitcoin. Nel contesto di applicazioni crittografiche chiamereo l'ingresso della funzione messaggio e lo indicheremo con la lettera M.

effetto valanga:

```
echo foo | time shasum -a 256
# output: b5bb9d8014a0f9b1d61e21e796d78dccdf1352f23cd32812f4850b878ae4944c
# tempi: 0.04 real 0.02 user 0.01 sys
echo fou | time shasum -a 256
# 3fdf270878a29979f66bff2d055147240d449e627f3d908e61ddf9b638c24a2a
# tempi: 0.04 real 0.02 user 0.01 sys
```

Alcune funzioni di hash notevoli In Bitcoin-core

SHASUM256 SHASUM256 è un algoritmo di hash che mappa un messaggio in un valore hash composto da 256 bits (32 byte). Nel codice bitcoin-core la classe responsabile per la generazione del valore di hash è CSHA256 (sha256.h)

L'ingresso è 'const unsigned char* data', L'uscita è 'const unsigned char hash[256]'

La classe CSHA256 è utilizzata dalla classe CHash256 che dato un 'const unsigned char* data' produce un 'const unsigned char hash[256]' applicando due volte CSHA256. In altri termini l'ingresso viene trasformato applicando due volte SHASUM256.

Infine una funzione (overload) inline hash256 produce un blob opaco di 256 bit utilizzando CHash256 a partire da oggetti e vettori.

Semplificando:

hash = SHASUM256(SHASUM256(data))

Questo doppio hash shasum256 prende il nome di shasum256d ed è ritenuto più sicuro contro l'attacco "extension lenght" (si veda "Practical Cryptography by Ferguson and Schneier"):

Why hashing twice?

Ciò nonostante va detto che l'argomento è controverso. Dunque non prenderemo, per il moento, posizione particolare.

The puzzle of the double hash

RIPEMD160 RIPEMD160 è un algoritmo di hash che mappa un messaggio in un valore hash composto da 160 bits (20 byte). Nel codice bitcoin-core la classe responsabile per la generazione del valore di hash è CRIPEMD160 (ripemd160.h)

L'ingresso è 'const unsigned char* data', L'uscita è 'const unsigned char hash[160]',

La classe CRIPEMD160 è utilizzata dalla classe CHash160 che dato un 'const unsigned char* data' produce un 'const unsigned char hash[160]' applicando CRIPEMD160.

Infine una funzione (overload) inline hash160 produce un blob opaco di 160 bit utilizzando CHash160 a partire da oggetti e vettori.

BECH32 Bech32 is a new bitcoin address format specified by BIP 0173. This page tracks the adoption of Bech32.

1.3 Firma digitale

1.3.1 Introduzione

La firma digitale è uno degli elementi fondanti di Bitcoin. Le firme consentono di validare la "proprietà di moneta", inviare e ricevere transazioni.

Si può paragonare la firma digitale ad una sorta di identità digitale. Al contario delle firme calligrafiche, le firme digitali sono fondate sulla crittografia; ciò significa che è praticamente impossibile produrre firme digitali false senza possederele chiavi private in grado di generarle.

Esistono molti schemi utilizzati per la creazione di firme digitali. Alcuni schemi di rilievo sono:

- RSA
- ECDSA
- Schnorr

In Bitcoin la firma digitale Elliptic Curve Digital Signature Algorithm ECDSA è uno degli strumenti fondanti per garantire la sicurezza. La firma digitale consente di firmare "le transazioni" per mezzo di una chiave privata segreta (privateKey). Altri soggetti possono verificare l'autenticità della firma per mezzo di una chiave pubblica (publicKey) associata alla privata per mezzo di relazioni matematiche. La chiave privata non deve essere disvelata e deve essere custodita con comportamenti sicuri. La chiave pubblica può essere distribuita senza compromettere la sicurezza. Va però segnalato che la distribuzione della chiave pubblica può comportare una perdita di privacy qualora ciò venga fatto senza accorgimenti volti a tutelare la privacy delle transazioni.

Nei paragrafi seguenti introdurremo progressivamente i concetti matematici utili alla comprensione della ECDSA.

Nel paragrafo [Transazioni](transazioni.md) viene descritto l'uso della firma digitale in Bitcoin.

Per comprendere il funzionamento della firma digitale in Bitcoin è necessario prendere confidenza con due argomenti:

- Campi Finiti (Algebra, Teoria dei Gruppi, Teoria dei Numeri, Aritmetica Modulare)
- Curve Ellittiche

La combinazione dei due da luogo alla firma digitale ECDSA.

Benché alcune operazioni non siano intuitive, per comprendere ad alto livello il funzionamento occorrono solo nozioni di base di matematica. La conoscenza della teoria dei gruppi e dell'analisi matematica costituiscono senza dubbio un valido aiuto per la comprensione.

Sapere le proprietà algebriche dei campi ci permette di comprendere la firma digitale.

1.3.2 L'insieme delle classi di resto modulo n

Sia $\mathbb{Z}$ l'insieme dei numeri interi e sia n un fissato intero positivo. Se $x, y \in \mathbb{Z}$ poniamo

$$x \equiv y \pmod{n}$$

se e solo se $\exists k \in \mathbb{Z} \mid x - y = k \cdot n$

La relazione $y \pmod{n}$, detta *congruenza di modulo n* è una relazione di equivalenza su $\mathbb{Z}$.

L'insieme quoziente, normalmente indicato con $\mathbb{Z}_n$, anziché con $\mathbb{Z}/_{\equiv}$, viene detto *l'insieme delle classi di resto modulo n.*

1.3.3 Gruppi

Definizione 1 (Gruppo)**.** Sia $\mathbb{G}$ un insieme e $\perp$ una operazione binaria interna su $\mathbb{G}$. Diremo che la struttura algebrica $(\mathbb{G}, \perp)$ è un *gruppo* se soddisfa le seguenti proprietà:

(G1.) l'operazione interna $\perp$ è associativa;

(G2.) $(\mathbb{G}, \perp)$ ammette elemento neutro;

(G3.) ogni elemento di G è invertibile;

Un gruppo $(\mathbb{G}, \perp)$ è detto *commutativo* o *abeliano* se soddisfa l'ulteriore proprietà:

(G4.) l'operazione interna $\perp$ è commutativa;

Solitamente, l'operazione di un gruppo viene indicata con il simbolo $+$ o con il simbolo $\cdot$.

Nel primo caso il gruppo $(\mathbb{G}, +)$ è detto *gruppo additivo*, l'elemento neutro è indicato con 0 e l'elemento inverso con $-x$.

Nel secondo caso il gruppo $(\mathbb{G}, \cdot)$ è detto *gruppo moltiplicativo*, l'elemento neutro è indicato con 1 è detto unità, l'elemento inverso di $x \in \mathbb{G}$ è indicato con x^{-1}.

> Si noti che dalle operazioni somma e moltiplicazione è possibile derivare la sottrazione e la divisione.
>
> Dato un gruppo additivo $(\mathbb{G}, +)$, la sottrazione in è detto è data dall'applicazione:
>
> $$- : \mathbb{G} \times \mathbb{G} \to \mathbb{G} \mid \forall x, y \in \mathbb{G}, x - y = x + (-y)$$
>
> Dato un gruppo moltiplicativo $(\mathbb{G}, \cdot)$, la divisione è data dall'applicazione:
>
> $$\div : \mathbb{G} \times \mathbb{G} \to \mathbb{G} \mid \forall x, y \in \mathbb{G}, x \div y = x \cdot y^{-1}$$

1.3.4 Anelli

Definizione 2 (Anello)**.** Sia $\mathbb{A}$ un insieme e $+, \cdot$ due operazioni binarie interne su $\mathbb{A}$ diremo che la struttura algebrica $(\mathbb{A}, +, \cdot)$ è un anello se soddisfa le seguenti proprietà:

(A1.) $(\mathbb{A}, +)$ è un gruppo abeliano;

(A2.) il prodotto è associativo;

(A3.) $\exists\ x, y \in \mathbb{A} \mid x \cdot (y + z) = (x \cdot y) + (x \cdot z) \wedge (y + z) \cdot x = (y \cdot x) + (z \cdot x)$

Un anello $(\mathbb{A}, +, \cdot)$ è detto *commutativo* se il prodotto è commutativo.
Un anello $(\mathbb{A}, +, \cdot)$ è detto *unitario* se il prodotto se $(\mathbb{A}, \cdot)$ ammette elemento neutro.

1.3.5 Corpi e campi

Definizione 3 (Corpo)**.** Dato un insieme $\mathbb{K}$ e due operazioni binarie interne su $\mathbb{K}$, diremo che la struttura algebrica $(\mathbb{K}, +, \cdot)$ è un *corpo* se soddisfa:

(K1.) $(\mathbb{K}, +)$ è un gruppo abeliano

(K2.) se 0 è l'elemento neutro di $(\mathbb{K}, +)$ e $\mathbb{K}^* = \mathbb{K} - \{0\}$, allora $(\mathbb{K}^*, \cdot)$ è un gruppo;

(K3.) il prodotto è distribuitivo rispotto alla somma.

Se inolte $(\mathbb{K}^*, \cdot)$ è abeliano, allora $(\mathbb{K}, +, \cdot)$ è detto *corpo commutativo* o *campo*

1.3.6 L'insieme delle classi di resto modulo p

L'aritmetica modulare si basa sul concetto di congruenza.

Dati due numeri interi a e b, si dicono congruente modulo p se la differenza (a - b) è multipla di p. In tal caso scriveremo:

$$a \equiv b \pmod{p}$$

Ad esempio,

$$\begin{aligned} 17 &\equiv 36 \pmod{19} \text{ perché } (36 - 17) = 19 \\ 11 &\equiv 49 \pmod{19} \text{ perché } (49 - 11) = 38 = 2 \cdot 19 \\ -1 + 17 &\pmod{19} = 18 \\ -2 + 17 &\pmod{19} = 19 \pmod{19} = 0 \\ -2 + 18 &\pmod{19} = 20 \pmod{19} = 1 \end{aligned}$$

In pratica altro che l'aritmetica dell'orologio alla quale tutti siamo abituati: 23:00 + 2 ore = ore 1:00. Nel caso dell'orologio il modulo è 12, nel caso dei campi finiti il modulo è un numero primo. Si tratta di una condizione sufficiente ad evitare che esistano divisori dello zero, ma andiamo per gradi.

Per indicare una classe di resto modulo p useremo $\mathbb{Z}_p$.

$\mathbb{Z}_p$ ha delle proprietà interessanti, tali da rendere la struttura algebrica composta da $\mathbb{Z}_p$ e la somma un gruppo abeliano (o gruppo commutativo):

- proprietà associativa: dati a, b, c appartenenti a $\mathbb{Z}_p$, (a + b) + c = a + (b + c)
- esistenza elemento neutro: dato a, l'elemento neutro della somma è lo 0, infatti a + 0 = a

- esistenza elemento inverso della somma: dato a, -a è l'elemento inverso, infatti a + (-a) = 0
- è commutativo: a + b = b + a

Anche la moltiplicazione gode delle stesse proprietà se escludiamo il numero 0 per il quale non esiste l'inverso:

- $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- elemento neutro: 1
- inverso: $a^{-1} = \frac{1}{a}$
- $a \cdot b = b \cdot c$

propietà distributiva!

Se con la somma non abbiamo problemi di sorta, con la moltiplicazione, ma soprattutto con la divisione cominciamo ad avere qualche difficoltà intuitiva.

Supponiamo:

$11 \cdot 7 = 77 = 4 \pmod{19}$

Quale è l'elemento inverso di 4? Ovvero quel numero che moltiplicato per 4 da 1? Sappiamo che $4 \cdot 5 = 20 \equiv 1 \pmod{19}$. Dunque, l'elemento inverso è 5, ovvero $4^{-1} = 5$. Non è molto intuitivo come processo e non è molto pratico andare per tentativi. Immaginiamo con grandi numeri.

Piccolo teorema di Fermat Per calcolare la divisione fra 1 e 4 ci viene in aiuto il piccolo teorema di Fermat:

$$a^{p-1} \equiv 1 \pmod{p} \tag{1.1}$$

dal quale, moltiplicando ambo i membri per a^{-1} si ricava:

$$a^{-1} \equiv a^{p-2} \pmod{p} \tag{1.2}$$

dunque, tornando al 4

$$4^{-1} = \frac{1}{4} = 4^{19-2} = 4^{17} = 17,179,869,184 \equiv 5 \pmod{19}$$

Il teorema funziona, ma già si cominciano ad intravedere problemi computazionali con la divisione nel in $\mathbb{Z}_p$.

Mentre moltiplicare è facile, invertire è molto difficile a livello computazionale. Questa difficoltà prende il nome di Problema del logaritmo discreto ed è un elemento cardine della ECDSA.

1.3.7 Campi Finiti

un *campo finito* $\mathbb{K}_p$ (p primo e $n \in \mathbb{Z} \mid n \geq 1$) è un campo che ha un numero finito di elementi tali che:

1. ogni campo finito ha p^n elementi, per qualche numero primo p e qualche numero naturale $n \geq 1$.
2. per ogni numero primo p e naturale $n \geq 1$, esiste un solo campo finito con p^n elementi, a meno di isomorfismi.

I campi finiti $\mathbb{K}_p$ sono delle strutture algebriche composte (per $n = 1$) da:

- l'insieme delle classi di resto modulo p $\mathbb{Z}_p$
- due operazioni binarie interne: la somma e la moltiplicazione
- generazione di una coppia chiave Privata, chiava Pubblica utilizzando ECDSA e secp256k1
- la chiave privata può firmare 'sign()' un messaggio
- la chiave pubblica può verificare che il messaggio sia stato firmato attraverso la chiave privata senza conoscerla:

1.3.8 Spazi vettoriali

1.3.9 Spazi affini

1.3.10 Spazi proiettivi

Definizione 4 (Spazio proiettivo)**.** Diremo *spazio proiettivo sul campo* $\mathbb{K}$ una terna $(V, \mathscr{P}, \theta)$, costituita da uno spazio vettoriale V su $\mathbb{K}$, da un insieme $\mathscr{P}$ e da una applicazione suriettiva $\theta : V - \{\mathbf{0}\} \to \mathscr{P}$, tale che:

(SP1.) $(\theta(\vec{u}) = \theta\vec{v}) \iff (\exists\ \alpha \in \mathbb{K}^* \mid \mathbf{w} = \alpha \cdot \mathbf{u})$

L'insieme $\mathscr{P}$ è detto sostegno dello spazio proiettivo $(V, \mathscr{P}, \theta)$ e i suoi elementi sono detti punti.

Definizione 5 (Dimensione)**.** Uno spazio proiettivo $\mathscr{P}$ sarà detto di dimensione finita se è associato ad uno spazio vettoriale V di dimensione finita. In tal caso porremo:

$$dim(\mathscr{P}) = dim(V) - 1$$

Esempio 1 (Spazio proiettivo standard $\mathbb{KP}^n = \mathscr{P}(\mathbb{K}^{n+1})$)**.** Lo spazio proiettivo canonicamente associato allo spazio vettoriale standard $\mathbb{K}^{n+1}$ sarà detto *spazio proiettivo standard di dimensione n sul campo* $\mathbb{K}$. Esso sarà solitamente indicato con $\mathscr{P}(\mathbb{K}^{n+1})$ o anche con $\mathbb{KP}^n$. Un punto di $\mathbb{KP}^n$ è, per definizione, una classe $P = [(x_0, \ldots, x_n)]$, dove $(x_0, \ldots, x_n)$ è una $(n+1)$-pla di elementi non tutti nulli di $\mathbb{K}$ e $(x'_0, \ldots, x'_n) \in [(x_0, \ldots, x_n)] \iff \exists\lambda \mid (x'_0, \ldots, x'_n) = \lambda(x_0, \ldots, x_n)$,

Esempio 2 (L'ampliamento proiettivo dello spazio affine standard $\mathbb{K}^2$)**.** Sia $\tilde{\mathcal{A}}^1 = \mathbb{K}$ la retta affine standard sul campo $\mathbb{K}$ e sia $\mathbb{KP}^0$ lo spazio proiettivo standard 0-dimensionale su $\mathbb{K}$, il cui unico punto sarà indicato con il simbolo $\infty \notin \mathbb{K}$. Sia poi

$$\theta : \mathbb{K}^2 - \{(0,0)\} \to \tilde{\mathcal{A}}^1(\mathbb{K}) \cup \{\infty\}$$

L'applicazione definita ponendo:

$$\theta(x^0, x^1) = \begin{cases} \infty, \text{ se } x^0 = 0 \\ \frac{x^1}{x^0}, \text{ se } x^0 \neq 0 \end{cases}$$

Dunque la terna $(\mathbb{K}^2, \tilde{\mathcal{A}}^1 \cup \{\infty\}, \theta)$ è una retta proiettiva sul campo $\mathbb{K}$, associata allo spazio vettoriale standard $\mathbb{K}^2$, detta *ampliamento proiettivo della retta affine standard* mediante l'aggiunta di un punto "improprio" o all'"infinito".

In maniera analoga può essere definito l'ampliamento proiettivo di una spazio affine di dimensione n: data la terna $(\mathbb{K}^{n+1}, \tilde{\mathcal{A}}^n \cup \mathbb{KP}^{n-1}, \theta)$ con

$$\theta : \mathbb{K}^{n+1} \to \tilde{\mathcal{A}}^n \cup \mathbb{KP}^{n-1}$$

definita ponendo:

$$\theta(x^0, \ldots, x^n) = \begin{cases} [(x_1, \ldots, x_n)], \text{ se } x^0 = 0 \\ (\frac{x^1}{x^0}, \ldots, \frac{x^n}{x^0}) \in \tilde{\mathcal{A}}^n(\mathbb{K}), \text{ se } x^0 \neq 0 \end{cases}$$

la terna $(\mathbb{K}^{n+1}, \tilde{\mathcal{A}}^n \cup \mathbb{KP}^{n-1}, \theta)$ è uno spazio proiettino n-dimensionale su $\mathbb{K}$ associato allo spazione vettoriale $\mathbb{K}^{n+1}$, detto amplimaneto proiettivo dello spazio affine standards $\tilde{\mathcal{A}}^n$, mediante l'aggiunta di un iperpiano (proiettivo) "improprio" o "all'infinito". Si noti che $\tilde{\mathcal{A}}^n \cup \mathbb{KP}^{n-1}$ può essere associato allo spazio proiettivo standard $\mathbb{KP}^n$ mediante la biiezione:

$$\tilde{\theta} : \mathbb{KP}^n \to \tilde{\mathcal{A}}^n \cup \mathbb{KP}^{n-1}$$

definita ponendo

$$\tilde{\theta}[(0, u^1, \ldots, u^n)] = [(u^1, \ldots, u^n)] \in \mathbb{KP}^{n-1} \wedge \tilde{\theta}[(1, x^1, \ldots, x^n)] = [(x^1, \ldots, x^n) \in \tilde{\mathcal{A}}^n(\mathbb{K})$$

ciò giustifica la locuzione: lo spazio proiettivo standard $\mathbb{KP}^n$ si ottiene aggiungendo un iperpiano (proiettivo) "improprio" allo spazio affine standard $\tilde{\mathcal{A}}^n$.

1.3.11 Curve Ellittiche

Un curva Ellittica E su un campo $\mathbb{K}$ è una curva non singolare di grado tre (cubica) nell'ampliamento proiettivo dello piano affine standard, costituita dalle coppie $(x, y) \in \mathbb{K}^2$ che soddisfano l'equazione affine:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1.3}$$

In particolare, essendo definita su un campo con caratteristica diversa da 2, 3l la (1.3) diviene:

$$y^2 = x^3 + a \cdot x^2 + b \tag{1.4}$$

dove la condizione di non singolarità è data da:

$$4a^3 + 27b^2 \neq 0 \tag{1.5}$$

$$\{(x, y) \in \mathbb{K}^2 \mid y^2 = x^3 + a \cdot x^2 + b \wedge 4a^3 + 27b^2 \neq 0\} \cup \{\mathbf{0}\} \tag{1.6}$$

Intersezione con una retta Se il campo soggiaciente una curva ellittica è algebricamente chiuso, ogni retta interseca sul **piano proiettivo** la curva ellittica in tre punti.

- un punto di tangenza va contato con molteplicità due;
- un punto di flesso con molteplicità tre;
- un punto all'infitito ha molteplicità uno;

Questa proprietà ci permetterà di definire una struttura algebrica con insieme soggiaciente i punti di una curva ellittica.

Lo schema impiegato ECDSA è basato su ECC *Elliptic Curve Cryptography* con curva ellittica **secp256k1**.

Questa curva ellittica ha equazione:

$$y^2 = x^3 + 7 \tag{1.7}$$

Se il dominio di applicazione della curva fosse quello dei numeri reali (ristretto al segmento $x|x^3 + 7 >= 0$) la curva avrebbe questo aspetto:

è facile comprendere perché sia simmetrica rispetto all'asse x perché suoi punti sono le radici di una cubica:

$$y = \pm\sqrt{(x^3 + 7)}$$

In ECC però il dominio della curva non è quello dei numeri reali o interi, ma è un **campo finito** $\mathbb{K}_p$ costruito con le classi di resto modulo p, dove p è un numero primo (detto anche campo di Galois).

Definizione di un gruppo tramite EC

Possiamo definire una struttura algebrica $(E(\mathbb{KP}^2), +)$ e dimostrare che si tratti di un gruppo:

- il supporto del gruppo è l'insieme dei punti della curva ellittica definita sul piano proiettivo $E(\mathbb{KP}^2)$;
- L'applicazione binaria interna additiva + è definita in questo modo: dati tre punti allineati P, Q e R (dati dall'intersezione della curva con una retta) la loro somma è tale che P + Q + R = 0;
- L'elemento neutro **0** è il punto all'infinito $E(\mathbb{KP}^1)$;
- L'elemento inverso di un punto P è l'elemento simmetrico rispetto all'asse x;

Con la somma si richiede che i tre punti siano allineati e tre punti sono allineati indipendentemente dall'ordine con i quali vengono presi.

$$P + (Q + R) = Q + (P + R) + R = R + (P + Q) = 0$$

In questo modo si dimostra sia la proprietà associativa che quella commutativa: siamo in un gruppo abeliano.

Come si computa la somma di due punti arbitrari?

Somma geometrica Grazie al fatto che valgono le proprietà precedenti possiamo scrivere:

$$P + Q + R = 0 \Rightarrow P + Q = -R$$

Questa equazione ci permette di definire, in modo geometrico, la somma di due punti P e Q:

- se disegniamo una linea che unisce i punti P e Q questa intersecherà la curva in un punto R
- prendendo l'inverso di R, -R, abbiamo ottenuto la somma di P e Q

Questo metodo geometrico funziona ma richiede alcune rifiniture. In particolare dobbiamo dare risposta alle seguenti domande:

* Cosa accade se P o Q uguagliano 0? (ovvero sono punti che proiettano all'infinito?) Avendo definito 0 come elemento identico abbiamo che P + 0 = P e 0 + Q = Q, per tutti i P e Q * Cosa accade se P = -Q? In questo caso P è l'inverso di Q, dunque P + Q = 0, per la definizione di elemento inverso. * Cosa accade se P = Q? In questo caso consideriamo la retta tangente alla curva ellittica in P e troviamo l'intersezione * Cosa accade se P e Q sono diversi ma non c'è un terzo punto R? Siamo in un caso simile al precedente e utilizziamo la tangente

Somma geometrica A questo punto abbiamo bisogno di un metodo algebrico per effettuare la somma

P e/o Q infiniti Se P = 0, P + Q = Q = R

dunque è immediato calcolare R

P e Q finiti e distinti Se P e Q sono distinti, la linea che li attraversa ha equazione:

m = (Yp - Yq) / (Xp - Xq)

L'intersezione con la curva assume la forma R:

$$Xr = m^2 - Xp - XqYr = Yp + m(Xr - Xp)$$

o, equivalentemente:

Yr = Yq + m (Xr - Xq)

dato che P + Q = -R =¿ (Xp, Yp) + (Xq, Yq) = (Xr, -Yr) simmetria asse x

P e Q finiti e distinti Questo caso va trattato differentemente. Le equazioni per Xr e Yr sono le stesse ma dato che Xp = Xq, dall'equazione

$$y^2 = x^3 + a \cdot x + b$$

otteniamo che

$$y_p = \pm\sqrt{x_p^3 + a \cdot x_p + b}$$

derivando per x otteniamo

$$m = (3Xp^2 + a)/(2Yp)$$

Moltiplicazione scalare Definiamo l amoltiplicazione scalare.

$$n \cdot P = \underbrace{P + P + \cdots + P}_{\text{n volte}}$$

Sembrerebbe che computare n P richieda l'esecuzione di n somma. Se n è composta da k digit, l'algoritmo avrebbe un complessità pari a $O(2^k)$. Un algoritmo migliore consiste nel raddoppiare e sommare. Supponiamo che n sia 151, la cui rappresentazione binaria è b10010111. Dunque n P può essere riscritto come

Partendo da destra otteniamo

* Prendiamo P * Prendiamo 2P * Sommiamo P e 2P * Prendiamo 2 (2P) * Raddopiiamo per ottenere 2^3P * Non sommiamo * Raddoppiamo 2^3P per ottenere 2^4P * Sommiamo al precedente * ... In questo modo usiamo 7 raddoppi e 4 addizioni

Dunque il nostro algoritmo diventa O(log n) = O (k)

Logaritmo Supponiamo di conoscere che Q = n P e di avere P e Q. Come computiano n? Questo problema prende il nome di problema logaritmico. Questo problema è un problema difficile "hard"

Applicazione in ECDSA Nel caso ECDSA il modulo è un numero primo p che, come si evince dal nome della curva stessa (secp256k) è un numero 256 bit che ha valore:

$p = 2^{256}–2^{32}–977 = 115792089237316195423570985008687907852837564279074904382605163141518161494337$

è prossimo a 2^{256}, benché non sia il numero primo più grande nell'intervallo 2^{256}.

Il generatore G è un punto su prodotto cartesiano Kp * Kp di coordinate:

x = 55066263022277343669578718895168534326250603453777594175500187360389116729240, y = 32670510020758816978083085130507043184471273380659243275938904335757337482424

La firma digitale con ECDSA (Elliptic Curve Digital Signature Algorithm) è possibile grazie ai campi finiti. In particolare, grazie all'esistenza del problema del logaritmo discreto che viene qui anticipato e la cui comprensione sarà più chiara più avanti nel corso del capitolo.

In un campo finito possiamo definire la moltiplicazione di un punto (coppia di coordinate x, y) per una quantità scalare. Dato un punto G(x, y) e uno scalre s, calcolare la moltiplicazione s G è facile:

$$P = s \cdot G$$

Ma noti G e P, calcolare la "divisione":

$$s = P \cdot G^{-1}$$

è infattibile a livello computazionale (nelle opportune condizioni che definiremo).

Ne consegue che se s è una chiave Privata (secreta, secret), e P una chiave Pubblica, non è possibile ricavare il segreto s nota $P = sG$.

In ECDSA firmare digitalmente un messaggio M significa applicare una funzione 'Sign' che a partire dal codice di hash del messaggio M genera una coppia di numeri interi r e s.

La funzione ha questa forma:

```
func Sign(
    rand io.Reader,
    priv *PrivateKey,
    hash []byte)
    (r, s *big.Int, err error) {
        // ...
}
```

fa uso di un generatore di numeri random [4], del puntatore a una chiave privata 'PrivateKey' e, ovviamente, del codice di hash di M.

La sicurezza della firma dipende da molti fattori, non ultima l'entropia del generatore di numeri random.

Il generatore di numeri random produce un numero k nell'intervallo [1, p-1].

- Questo numero viene utilizzato per ricavare un punto random R = k * G sulla curva, r = R.x - $s = k^{-1} * (h + r * privKey)(modp) => s * k = h + r * PrivKey$ - ritorna r, s

The private key is a random integer chosen from (where is the order of the subgroup). The public key is the point where is the base point of the subgroup).

Firma di Schnorr

Gli sviluppatori bitcoin stanno proponendo l'adozione della firma di Schnorr. I vantaggi derivanti dal possibile utilizzo dello schema Schnorr sono i seguenti [3]:

- Sicurezza provabile: le firme Schnorr sono provabilmente sicure.

[4] Si presti particolare attenzione a questo fatto. Nel paragrafo relativo alla malleabilità vi si farà riferimento

- Non malleabilità
- Linearità

1.4 Indirizzi Bitcoin

¡img src=”./img/CPKDSS@2x.jpg” alt=”bitcoin-address-generation” width=”100

1.5 Transazioni

1.5.1 Hash di una transazione

1.5.2 Albero di Merkle

1.5.3 UTXO

1.5.4 Malleabilità delle transazioni

In precedenza (1.3.11) abbiamo visto che la firma digitale ECDSA è inerentemente malleabile. Ciò conduce al problema della malleabilità delle transazioni. Malleare una transazione significa malleare la firma (le firme) in maniera tale da mantenere valida la transazione in sé; il contenuto della transazione non può essere modificato senza invalidare la transazione stessa, ma se la firma viene malleata, pur mantenendosi valida, l'hash della transazione cambia e così il suo txid.

1.5.5 Segwit

1.6 La catena

1.7 Blocco

Le informazioni che riguardano la struttura di un blocco sono contenute nei file bitcoin-core primitives/block.h e primitives/block.cpp. Come si legge nel file, i nodi verificano e collezionano le transazioni dentro un blocco (con le logiche che vedremo) e ricavano un hash complessivo delle stesse attraverso una struttura dati che prende il nome di albero di Merkle. Dunque risolvono l'algoritmo di consenso *Proof of Work* che consiste nella ricerca iterativa di un valore nonce 'nNonce' da inserire nella testata del blocco (insieme alle altre informazioni) e che dia luogo a un hash di blocco inferiore ad un numero dato che prende il nome di *target*. Il target è collegato alla [difficoltà](./difficolta.md) 'nBits'. Risolto questo algoritmo trasferiscono il blocco agli altri nodi e tutti, se valido, lo inseriscono nella blockchain. La prima transazione del blocco è una transazione speciale che prende il nome di *coinbase*. Si tratta di una transazione che conia moneta a ricompensa del lavoro svolto il nodo che ha trasmesso per primo il nuovo blocco valido. Questa procedura è competitiva: tutti i nodi minatori costruiscoo il blocco destinando la transazione coinbase a un destinatario di loro scelta (in genere il minatore stesso). ¿ Dopo [SegWit](segregated-witness.md) l'albero di Merkle contiene anche script e firme tramite l'innesto di un sotto-albero per tramite della transazione coinbase.

1.7.1 Hash di un blocco

1.8 Consenso

1.8.1 Difficoltà

1.8.2 Miner

1.8.3 Sussidio

1.8.4 Commissioni

1.8.5 In codice

Mining Bitcoin-core esegue la PoW in mining.cpp all'interno della funzione generateBlocks.

```
// permette di inizializzare il nuovo blocco prima della PoW con
void IncrementExtraNonce(CBlock* pblock, const CBlockIndex* pindexPrev, unsigned
    int& nExtraNonce)
{
  //...
}
```

Poi con un ciclo while viene modificato solo il nounce

```
while (nMaxTries > 0 && pblock->nNonce < std::numeric_limits<uint32_t>::max() && !
    CheckProofOfWork(pblock->GetHash(), pblock->nBits, Params().GetConsensus()) &&
     !ShutdownRequested()) {
    ++pblock->nNonce; // qui
    --nMaxTries;
}
```

GetNextWorkRequired è chiamata da UpdateTime chiamato da CreateNewBlock In maniera analogo dobbiamo modificare la PoW per evitare di inizializzare i dati ad ogni step. In analogia sposterò generateBlock in mining. Al momento MineTx in Network

Rappresentazione della difficoltà In Bitcoin, il campo nBits **codifica** la difficoltà D della Proof of Work del blocco. Non va confuso con la difficoltà D alla quale è legato dalla relazione (2.4) che riformulata diviene:

$$T = \frac{T_{max}}{D} \tag{1.8}$$

Il valore **target** è rappresentato in formato Compat, simile ai numeri in virgola mobile IEEE754, e occupa 32 bit della testata.

Esponente (b)	Segno (s)	Mantissa (a)
8 bit [31-24]	1 bit [23]	23 bits [22-00]

Per convertire questo valore in formato intero si utilizza la seguente formula:

$$T = (-1)^s \cdot a \cdot 256^b \tag{1.9}$$

Conversione da nBits a Target La procedura per computare T è la seguente:

- ricavare la mantissa tramite i primi 23 bits: nBits & 0x007fffff
- ricavare il segno: nBits & 0x00800000 != 0
- ricavare l'esponente shiftando a destra 24 bit: nBits $\gg 24$
- se $b \leq 3$: translare a destra di $8 \cdot (3 - b)$ bit la mantissa
- se $b > 3$: translare a sinistra di $8 \cdot (b - 3)$ bit la mantissa
- applicare il segno

Esempio Ad esempio il valore di nBits del blocco di genesi vale nBits = 0x1d00ffff da cui:

$$\begin{aligned} a &= \text{0x1d00ffff} \ \& \ \text{0x007fffff} = \text{0xffff} \\ s &= (\text{0x1d00ffff} \ \&\text{0x00800000!} = 0) = 0 \\ b &= \text{0x1d00ffff} \gg 24 = \text{0x1d} > 3 \\ a &= a << 8 \cdot (b - 3) = \text{0xffff} << \text{0xd0} = \text{0xffff} \ll 208 \\ a &= (-1)^0 \cdot a = \text{0xffff} \ll 208 \end{aligned} \tag{1.10}$$

Il valore del target massimo, in esadecimale, è il seguente:

$T_{max} = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF$ (56 volte F = $\frac{224}{\ln(16)}$),

che, a causa della rappresentazione compatta, diviene:

$T_{max} = 0xFFFF0000000000000000000000000000000000000000000000000000000000000000$ (4 volte F e 52 volte 0).

Conversione da Target a nBits Il valore nBits è settato dal metodo unsigned int *GetNextWorkRequired()* che, sotto le opportune condizioni, invoca *GetCompact()*.

Come si vede dal codice sotto riportato si tratta della procedura inversa.

```
// arith_uint256
template <unsigned int BITS>
unsigned int base_uint<BITS>::bits() const
{
    for (int pos = WIDTH - 1; pos >= 0; pos--) {
        if (pn[pos]) {
            for (int nbits = 31; nbits > 0; nbits--) {
                if (pn[pos] & 1U << nbits)
                    return 32 * pos + nbits + 1;
            }
            return 32 * pos + 1;
        }
    }
    return 0;
}

uint32_t arith_uint256::GetCompact(bool fNegative) const
{
    int nSize = (bits() + 7) / 8;
    uint32_t nCompact = 0;
    if (nSize <= 3) {
        nCompact = GetLow64() << 8 * (3 - nSize); // qui viene impostata la
    mantissa
```

```
23      } else {
24          arith_uint256 bn = *this >> 8 * (nSize - 3);
25          nCompact = bn.GetLow64(); // qui viene impostata la mantissa
26      }
27      // The 0x00800000 bit denotes the sign.
28      // Thus, if it is already set, divide the mantissa by 256 and increase the
        exponent.
29      if (nCompact & 0x00800000) {
30          nCompact >>= 8;
31          nSize++;
32      }
33      assert((nCompact & ~0x007fffff) == 0);
34      assert(nSize < 256);
35      nCompact |= nSize << 24; // qui viene impostato l'esponente
36      nCompact |= (fNegative && (nCompact & 0x007fffff) ? 0x00800000 : 0); // qui
        viene impostato il segno
37      return nCompact;
38  }
```

1.8.6 Portafoglio

1.9 Rete

1.9.1 DNS e indirizzi

1.9.2 Protocollo

1.10 Sviluppi in corso

lo sviluppo di Bitcoin verte su due aspetti principali:

- La scalabilità
- La privacy

Parte II

Metodi

Capitolo 2

Calcolo della profittabilità del *mining*

2.1 Obbiettivi

This note shows a simple method to compute the expected value of the coin-base subsides gainable competing as a miner in the Bitcoin network. In particular, we inspect the profitability of running ASIC devices summing to an amount of "hash rate" h.

Many web sites, to compute the profitability, use methods that can be misleading because they don't take into account the global hash rate growth of the Bitcoin network.

2.2 Trattazione della difficoltà

La funzione di hash SHASUM256 è una applicazione che mappa dati interi non negativi in un intero appartenente all'intervallo chiuso $\left[0, 2^{256} - 1\right]$:

$$h : \mathbb{Z}^+ \cup 0 \rightarrow \{z \in \mathbb{Z}^+ \cup 0 \mid z \leq 2^{256} - 1\} \tag{2.1}$$

In Bitcoin, per produrre un blocco valido, è necessario formare una testata di blocco tale per cui l'hash della stessa dia luogo a un codice h_{block} che rispetti la condizione:

$$h_{block} < T \tag{2.2}$$

dove T è il valore target (*bnTarget* in pow.cpp). In Bitcoin la difficoltà corrente è indicata nel campo nBits della testata del blocco. Il formato del campo nBits è Compact, maggiori informazioni sono in appendice.

La difficoltà di questa ricerca è dovuta al fatto che gli hash generati nel corso della *Proof of Work* sono distribuiti uniformemente nell'intervallo $\left[0, 2^{256} - 1\right]$ ma il valore target T è molto inferiore a $2^{256} - 1$.

Per scelta progettuale T_{max}, il massimo valore che T può assumere, vale $2^{224} - 1$, e dunque:

$$T \leq T_{max} = 2^{224} - 1 \ll 2^{256} - 1 \tag{2.3}$$

La difficoltà D è definita nel file blockchain.cpp e è data dalla seguente espressione:

$$D = \frac{T_{max}}{T} \tag{2.4}$$

La ricerca di un hash valido è una **prova di Bernoulli** che ha probabilità di successo:

$$p(h_{block}^{(i)} < T) = \frac{T}{2^{256}} = \frac{T_{max}}{D \cdot 2^{256}} = \frac{1}{2^{32} \cdot D} \tag{2.5}$$

dove i rappresenta l'i-esimo tentativo.

Dalla (2.5), si evince che la massima probabilità possibile è data da $D = 1$ e vale $p = 2^{-32}$.

La distribuzione di probabilità del numero di tentativi k necessari per per ottenere un hash valido è data dalla distribuzione geometrica:

$$P(X = k) = (1 - p)^{k-1} \cdot p^1 \tag{2.6}$$

per $k \in \mathbf{Z}^+$

La distribuzione geometrica ha valore atteso e varianza date dalle seguenti espressioni

$$\mathbb{E}(X) = p^{-1} = \frac{2^{256}}{T} \tag{2.7}$$

Dal quale si ricava che il numero medio di tentativi necessari è molto ampio ed ha varianza $\approx \frac{1}{p^2}$:

$$\sigma^2(X) = \frac{1 - p}{p^2} \tag{2.8}$$

In data odierna, i valori sono i seguenti:

$$\begin{cases} D = 1.55 \cdot 10^{13} \\ p = 1.51 \cdot 10^{-23} \\ \mathbb{E}(X) = 6.64 \cdot 10^{22} \\ \sigma^2(X) = 4.41 \cdot 10^{45} \end{cases} \tag{2.9}$$

Un processo di arrivi di Poisson è un processo stocastico utile per modellare gli arrivi in un sistema. Può essere visto come la versione continua di un processo di Bernoulli. Per un processo di Bernoulli gli arrivi possono verificarsi soltanto in corrispondenza in multipli interi di un incremento dato.

In un processo di Poisson gli arrivi possono verificarsi arbitrariamente e la probabilità di un arrivo per ogni particolare istante è nulla. Ciò significa che non esiste un modo per descrivere la probabilità di arrivo in un punto ed è più facile riferirsi a sequenze di arrivi in un intervallo; tali sequenze sono quantità i.i.d.

In letteratura la ricerca *Proof of Work* è spesso ritenuta modellabile con sufficiente approssimazione attraverso un processo di Poisson di parametro λ.

Se la variabile casuale X rappresenta il numero di arrivi in un intervallo di tempo t, indipendente dall'origine dei tempi, sotto le le ipotesi:

- uniformità
- indipendenza e assenza di memoria
- ordinarietà

la densità si distribuisce come una variabile casuale di parametro $\lambda = r \cdot t$.

$$r = \frac{h}{2^{32}D} \tag{2.10}$$

$$\lambda = \frac{h}{2^{32}D} \cdot t \tag{2.11}$$

dove h è la frequenza di hash ($\frac{H}{s}$) e λ è il valore atteso degli arrivi nell'unità di tempo:

$$\mathbb{E}_1[X] = \lambda \tag{2.12}$$

la varianza della distribuzione è la seguente:

$$\mathbb{E}_1[(X - \mathbb{E}_1[X])^2] = \sigma^2 = \lambda \tag{2.13}$$

La distribuzione di probabilità (PDF) ha la seguente forma:

$$f(X;t) = \lambda \cdot e^{-\lambda} = rt \cdot e^{-rt} \tag{2.14}$$

dove t è la durata della batteria di prove espresso in secondi (s).

La distribuzione di probabilità cumulativa (CDF) è data da:

$$P(X;t) = 1 - e^{-rt} = 1 - e^{-\lambda} \tag{2.15}$$

2.2.1 Esempio

Se $h = 1\ GH/s$ e $D = 1690906$ la remunerazione per ogni arrivo e $t = 86400\ s$ (1 giorno), la probabilità di arrivo in un giorno è pari a:

$$\lambda = \frac{h \cdot t}{2^{32} \cdot D} = \frac{1 \cdot 10^9 \cdot 86400}{2^{32} \cdot 1690906} \approx 0.0119 \tag{2.16}$$

Se $S_B = 12.5\ BTC$ BTC, possiamo ricavare il sussidio utilizzando dalla seguente:

$$S = \lambda \cdot S_B \tag{2.17}$$

che nel caso in esame è pari a 0.59 BTC (i valori attuali sono molto distanti da questo semplice esempio!).

La varianza del processo è ancora λ, dunque 0.0119, e la deviazione standard $\sigma = 0.109$, circa 9.17 volte superiore.

Va sottolineato che il processo è random e senza memoria.

Imponendo la (2.15) a 0.8 otteniamo il seguente valore per t:

$$P(X;t) = 1 - e^{-\lambda \cdot t} = .80 \Rightarrow e^{-\lambda \cdot t} = 0.20 \Rightarrow \lambda \cdot t = -\ln(0.20) \Rightarrow t \approx 135 giorni! \tag{2.18}$$

2.3 Stima dell'hash globale del network

Il protocollo Bitcoin adatta il valore della difficoltà D ogni 2016 blocchi in maniera tale da inseguire il valore 1 blocco ogni 600 secondi.

Approssimando, è dunque possibile ricavare una stima della frequenza di hash della rete H.

$$\lambda_{net} \cdot t = \frac{H}{2^{32} D} \cdot t = 1 \Rightarrow H = \frac{2^{32} \cdot D}{t} \tag{2.19}$$

per $t = 600$ sec

2.4 Riduzione della varianza mediante la partecipazione in un Pool

La partecipazione ad un pool di mining di "potenza" $h_{pool} = k \cdot h$ induce una varianza σ^2_{pool} k volte inferiore.

Se ad esempio $K = 100$, pur mantenendo costante il valore atteso pari a 0.59 BTC, la P_{pool} diviene

$$t = -\frac{\ln(0.20)}{k \cdot \lambda} \approx 1.35 \ giorni! \tag{2.20}$$

La varianza e ridotta proporzionalmente a k.

Il gestore del pool remunera la propria attività trattenendo una quota f del sussidio. Gli schemi adottati sono vari, ed includono la trattazione delle commissioni. In questa parte assumeremo l'uso della schema PPS e trascureremo, per il momento, le parte reltiva alle commissioni.

Alcuni degli schemi adottati da alcuni pool possono essere consultati.

In questo scenario dovremo tenere in considerazione che il ricavato dall'attività di mining, beneficiando della riduzione di varianza, subirà una riduzione:

$$\mathbb{E} = (1 - f) \cdot \frac{h \cdot t}{2^{32} \cdot D} \tag{2.21}$$

```
/* Example */
var k = 1000; // partecipanti pool
var P = .95;
var D = 15466098935554.65;
var day = 86400; // seconds
var h = 73 * Math.pow(10, 12);
var lambda = k * h / (Math.pow(2, 32) * D);
console.log("lambda pool", lambda)
var p = lambda * day;
console.log("p pool(1 day) = ", p)
var t = - Math.log(1- P) / lambda / (day);
console.log("P = ", P, "=> time = ", t, "days")
console.log("pool pay out day", p * 25 * 10000)
var BTCUSD = 9500
var b1 = p * 25 * BTCUSD / k
console.log("pay out day", b1)
var e = 24 * 2.9 * 0;
var c1 = e + 2000/365;
console.log("costs", c1)
console.log("profit per kWh", (b1 - c1)/24/2.9)
```

2.5 Hash rate growth

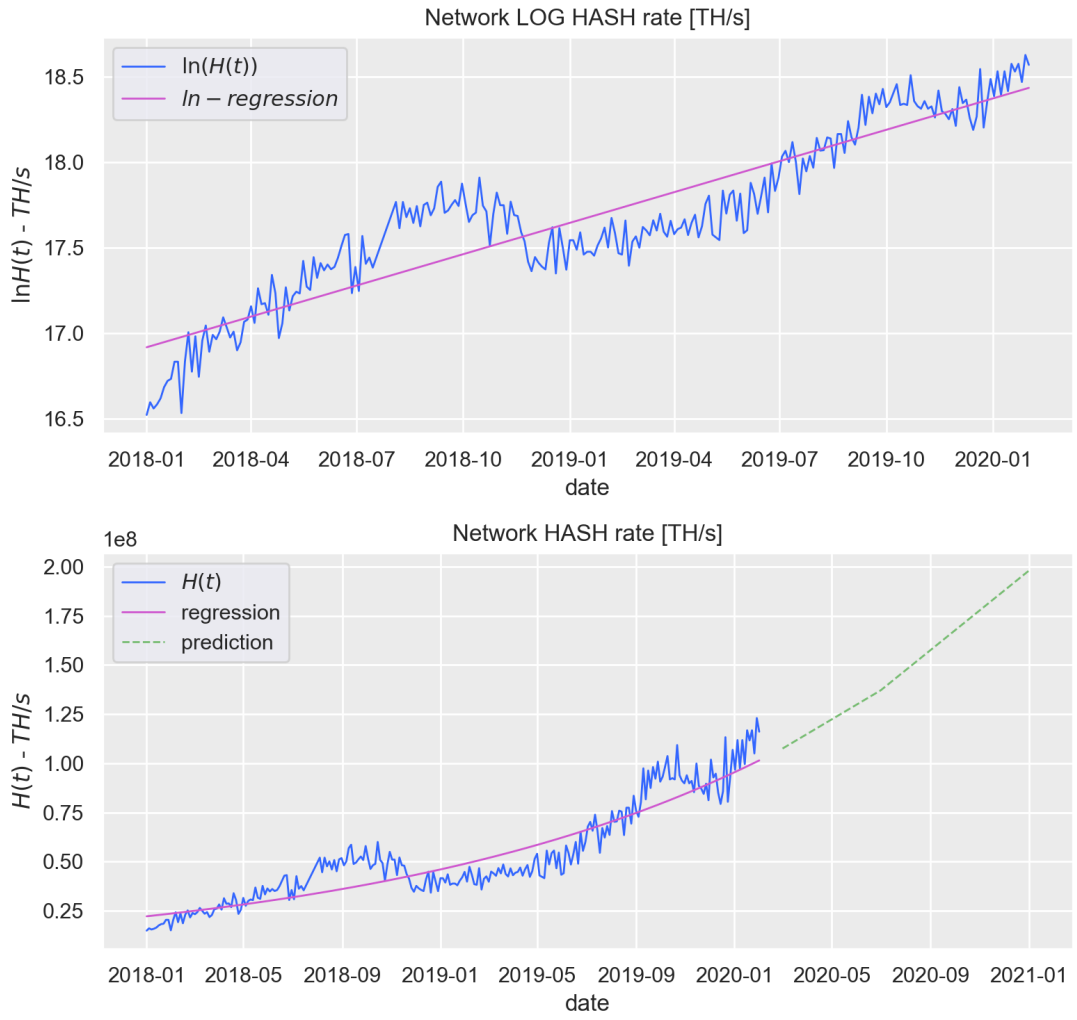

Figura 2.1: Global log-hash rate of the Bitcoin network from Jan, 2018 to Feb, 2020.

The global hash rate can be esteemed indirectly through the difficulty that is recorder in the block header:

$$\widehat{H_i} = \frac{2^{32}}{600} \cdot D_{i+1} \tag{2.22}$$

We have downloaded the estimated network hash rate fome the website blockchain.com.

The Figure (2.1) shows the natural logarithm of the hash rate growth $\ln(H(t))$ from Jan 1, 2018 to Feb 1, 2020.

The time unit is seconds (s), and the format is the unix epoch, thus seconds elapsed from January 1, 1970.

The hash rate unit is tera hashes per seconds (TH/S).

Given the data, we have fit the data to a line using the log-linear regression:

$$\ln(\widehat{H}(t)) = m \cdot t + q \tag{2.23}$$

from which we have derived:

$$\widehat{H}(t) = e^{q} \cdot e^{m \cdot t} \tag{2.24}$$

Equation (2.24) serves as a **short term predictor** for future global hash rate values.

The regression exhibits an acceptable fitness to the data. In the future developments of this method, we will show errors and residuals.

We know that growth might rise or decline, but the behaviour is reasonably log-linear since 2014.

Fitting the data, the parameters of the equation (2.24) are the following values:

$$\begin{cases} m \approx +2.29 \cdot 10^{-8} \\ q \approx -17.82 \end{cases} \tag{2.25}$$

2.6 Probability of winning a block

The Bitcoin protocol is coded to adjust the difficulty in such a way that the expected time to build a block is 10 minutes (600 seconds).

So we can imagine the rent-seeking competition among miners ([1]) to build a new block as composed of rounds of 10 minutes each.

Given the global network hash rate $H(t)$ and the hash rate owned by a single miner (or mining pool) $h(t)$, we can compute the probability $p_{win}(t)$ of gaining the subside S_{block} for a single round $r(t)$ that starts at t and ends at $t + 600$ seconds under the assumption that all the competitors use the "same technology" and the "same algorithms".

The same algorithms assumption seems to be accepted as guaranteed by the Bitcoin protocol and by the nature of the Proof of Work and its cryptographic hash functions (the underlying probability distribution is uniform) that entitles us to consider the random variables independent and identically distributed.

The same technology assumption is weak and serves to overcome some issues related to networking and other minors and majors issues.

The probability is:

$$p_{win}(t) = \frac{h(t)}{H(t)} \tag{2.26}$$

In our scenario, $h(t)$ is constant in time and it is much lower than $H(t)$:

$$h(t) = h_o \ll H(t) \tag{2.27}$$

So we can write:

$$p_{win}(t) = \frac{h(t)}{H(t)} = \frac{h_0}{H(t)} \tag{2.28}$$

2.7 Expectation

We can define a random variable X such that:

$$X = \left\{ \begin{array}{l} X = 1, \textit{ if the miner wins the round} \\ X = 0, \textit{ if the miner loses the round} \end{array} \right\} \tag{2.29}$$

The expected value of X over n rounds is:

$$\mathbf{E}(X) = \sum_{r=1}^{n} 1 \cdot p_{win}(t(r)) + 0 \cdot (1 - p_{win}(t(r))) \tag{2.30}$$

Where:

$$t(r) = t_{(r=1)} + 600 \cdot r \tag{2.31}$$

Rearranging the preceding equations we can write:

$$\mathbf{E}(X) = \sum_{r=1}^{n} \frac{h0}{\widehat{H}(t(r))} \tag{2.32}$$

Given equation (2.32) the following code follows:

```
# ------------------------------------------------
# Compute the expected value of X
# ------------------------------------------------

E = 0
r = 0

while(r <= 365 * 24 * 6): # e.g. 1 year
    r += 1
    E += h0 / H(t(r))

return E
```

Or, if we want to compute the expected number of rounds needed to win a block:

```
# ------------------------------------------------
# Compute the number of rounds needed to win a round
# ------------------------------------------------

E = 0
r = 0

while(E < 1): # e.g. 1 win
    r += 1
    E += h0 / H(t(r))

return r
```

2.8 Results

Given a single high-end commercial ASIC device, the hash rate is $h_0 = 73\ TH/s$ and under our assumptions the expected number of wins in a year, starting from Feb. 2, 2020, is:

$$\mathbf{E}(X) = \sum_{r=0}^{365*24*6} p_{win}(t(r)) \approx 0.026 \tag{2.33}$$

Obviously, given $N = 100$ devices of the same model, the result is:

$$\mathbf{E}(X) = \sum_{r=0}^{365*24*6} p_{win}(t(r)) \approx 2.58 \tag{2.34}$$

2.9 Economic prospects drafts

In this paragraph, we show two simplified economic prospects.

Given the exhibited exponential growth of the global hash-rate, and given the fact that ASIC devices are not repurposable, we have decided to take into account all the device cost without amortization.

Given the BTC price fluctuation (volatility), we have also avoided taking account of interests rates.

```
Rounds           : 52560 ( 365 ) days
-------------------------------------------------------
Device hash rate : 73 TH/s
Expeted wins     : 0.0269128246059
Subside  and fees: 12.55 BTC
Expected BTCs    : 0.337755948804
BTC/USD          : 9450 USD
Expected Value   : 3191.7937162 USD
Energy consum.   : 25404.0 kWh (28.95 W/TH/s)
Energy price.    : 0.1 USD/kWh
Energy cost      : 2540.4 USD
Device cost      : 2000 USD
                   ----------
Total costs      : 4540.4
Expected Profit  : -1348.6062838
         per kWh : -0.0530863755237
```

Figura 2.2: Model results for $S = 12.5BTC$

```
Rounds           : 52560 ( 365 ) days
-------------------------------------------------------
Device hash rate : 73 TH/s
Expeted wins     : 0.0269128102299
Subside  and fees: 6.3 BTC
Expected BTCs    : 0.169550704448
BTC/USD          : 9450 USD
Expected Value   : 1602.25415704 USD
Energy consum.   : 25404.0 kWh (28.95 W/TH/s)
Energy price.    : 0.1 USD/kWh
Energy cost      : 2540.4 USD
Device cost      : 2000 USD
                   ----------
Total costs      : 4540.4
Expected Profit  : -2938.14584296
         per kWh : -0.115656819515
```

Figura 2.3: Model results for $S = 6.25BTC$

2.9.1 Simulazione arrivo blocchi

Di seguito analizzeremo le relazioni che ci contentiranno di costruire un simulatore. Per simulare un processo futuro non possiamo fare ricorso alla stima sperimentale del valore di H, ma faremo uso della funzione di crescita esponenziale $\hat{H}$ che ci consentirà di predirre valori futuri a breve termine.

Processo di Poisson

Assumendo che l'arrivo di hash validi segua un processo di Poisson (riferimenti), chiamiamo r il numero di eventi nell'unità di tempo:

$$r = \frac{h}{2^{32} \cdot D}$$

λ è dunque data dalla relazione:

$$\lambda = r \cdot t = \frac{h \cdot t}{2^{32} \cdot D} \tag{2.35}$$

la densità di probabilità segue la legge:

$$\mathbb{P}(k \textit{ eventi in } t) = \frac{(r \cdot t)^k \cdot e^{-r \cdot t}}{k!}$$

in particolare per $k = 1$ (1 evento):

$$\mathbb{P}(1 \textit{ evento in } t) = (r \cdot t) \cdot e^{-r \cdot t} \tag{2.36}$$

il valore medio e la varianza equivalgono a λ:

$$\mathbb{E}(X) = \sigma^2(X) = r \cdot t = \lambda$$

da cui

$$\sigma(X) = \sqrt{\lambda}$$

Attraverso il metodo proposto da Knuth possiamo generare eventi che seguono la distribuzione:

$$\mathbb{P}(T_1 \leq t) = 1 - e^{-r \cdot t} \Longrightarrow e^{-r \cdot t} = (1 - \mathbb{P}) \Rightarrow t = -\frac{\log_e(1 - \mathbb{P})}{r}$$

Trattazione dinamica semplificata

La frequenza di hash globale della rete è dinamica. Durante il k-esimo round, assumendo i intero positivo tale che $i = \frac{k}{2006}$, per quanto codificato dal protocollo possiamo scrivere:

$$\bar{H}_i = \frac{2^{32}}{600} \cdot D_{i+1}$$

sostituendo nella (2.35):

$$\bar{\lambda}_i = r_i \cdot t = \frac{D_{i+1}}{D_i} \cdot \frac{t}{600}$$

D_{i+1} è calcolato dopo 2016 blocchi per tentare di imporre $\lambda_{i+1} \cdot T_{(i+1)}^{(2016)} = 2016$

Dunque assumendo una legge dinamica per $\bar{H}(t)$, desunta dai valori storici, possiamo ricavare λ e simulare il processo degli arrivi.

Vogliamo trovare un valore λ che ci consenta di simulare la batteria i-esima. Sappiamo con certezza che nel corso della i-esima batteria D non varia, ma H sì. Duqnue dobbiamo trovare un lambda adeguato a decrivere la densità di probabilità nell'intervallo considerato.

Per semplicità ci riferiremo ad una intera batteria.

Utilizzeremo il predittore $\hat{H}$ ricavato nel paragrafo (2.3) per simulare una serie di batterie di mining.

Possiamo riformulare la (2.24) in questo modo

$$\hat{H}(t) = e^{m \cdot t + q} = e^q \cdot e^{m \cdot t} = Q \cdot e^{m \cdot t}$$

Una batteria di round inizia all'istante t_0 e termina in $t_0 + T_{2016}$. T_{2016} corrisponde all'istante di arrivo del blocco numero 2016 (di batteria). Per costruzione, T_{2016} approssima T_{2W} che è pari a 2 settimane corrispondenti a $14 \cdot 24 \cdot 60 \cdot 60 = 1,209,600$ secondi. Ciononostante, in ragione della dinamicità di $H(t)$, che pur se **non monotona** tende a crescere, T_{2016} tende ad essere inferiore a T_{2W}. Al termine della batteria, il codice re-imposta la difficoltà utilizzando il fattore moltiplicativo:

$$k = \frac{T_{2W}}{T_{2016}}$$

in maniera tale che

$$D_{i+1} = k \cdot D_i$$

Dalla quale si ricava il rapporto

$$k = \frac{T_{2W}}{T_{2016}} = \frac{D_{i+1}}{D_i} = \frac{\bar{H}_i}{\bar{H}_{i-1}}$$

dunque

$$k = \frac{\hat{H}(t_0 + T_{2016})}{\hat{H}(t_0)}$$

attraverso la quale ricaviamo

$$\frac{Q \cdot e^{m\cdot(t_0+T_{2016})}}{Q \cdot e^{m\cdot(t_0)}} = \frac{Q \cdot e^{m\cdot(t_0)}}{Q \cdot e^{m\cdot(t_0)}} \cdot e^{m\cdot T_{2016}} = e^{m\cdot T_{2016}} = \frac{T_{2W}}{T_{2016}}$$

$$e^{m\cdot T_{2016}} = \frac{T_{2W}}{T_{2016}}$$

da cui

$$m = \frac{1}{T_{2016}} \ln(\frac{T_{2W}}{T_{2016}})$$

riformulando

$$T_{2016} \cdot e^{m\cdot T_{2016}} - T_{2W} = 0 \tag{2.37}$$

Se poniamo $T_{2016} = 2016 \cdot T_{mean}$, possiamo trovare una espressione per il tempo medio di mining di un blocco:

$$e^{m\cdot 2016\cdot T_{mean}} = \frac{600}{2016 \cdot T_{mean}}$$

Per convenienza utilizzeremo il metodo di Newton–Raphson per risolvere numericamente la (2.37) precedente, ricavandone un valore approssimativo per T_{2016}

$$f(x) = x \cdot e^{mx} - T_{2W} \tag{2.38}$$

A questo punto, avremo a disposizione r per la batteria in corso:

$$\hat{r} = \frac{2016}{T_{2016}}$$

e questo sarà il valore che utilizzeremo per generare gli eventi del processo di Possion.

Si noti che:

$$\int_{t_0}^{t_0+T_{2016}} \hat{H}(t)\ dt = \bar{H} \cdot T_{2016} \tag{2.39}$$

Trattazione con processo non omogeneo

$$\tilde{\Lambda}(\tau, t) = \int_0^t r(t) d\tau \tag{2.40}$$

Assumendo

$$\tilde{\Lambda}(\tau, t) = \int_{t_0}^{t_0+T_{2016}} r_0 \; e^{mt} \; d\tau \tag{2.41}$$

Integrare

2.10 Conclusion

Many input variables condition the profitability of the mining activity.

For example, the BTC price and fees, the global hash-rate growth, the energy and devices costs, and so on. Further, we should take into account network costs and others.

So, we should refine this method, but it serves as a starting point to debate and improve current public available practices.

This note highlight that within the assumptions and the given costs, solo-mining (or participating in a small pool) is hardly-profitable and poses some questions that we would further investigate:

- The cost of electric energy has a high impact on profitability. In which condition we might reduce energy costs?
- To take account of other factors that have an impact on the marginal utility (e.g. privacy)
- Is the ASIC market overrated? Considering that the devices are not repurposable and the obsolescence rate is very high, the current commercial price seems unjustified.
- The system seems to be under the pressure of centralizing forces (both geographically and economically). What will change in the next future? The BTC price? The hash-rate growth? What else? How?

With this note, we would give publicity to the fact that the end-user market needs a better model to compute profitability.The proposed method is a point where to start reasoning.

Capitolo 3

Analisi SVD

Supponiamo di disporre di n *ticker* ordinati. Vogliamo computare il numero di campioni $k \in \mathbb{Z}$ che si possono estrarre attraverso un finestra w che scorre di un numero di step pai a "s".

$$k \cdot s + w = n \Rightarrow k = \frac{n - w}{s}$$

Dunque se $n = 10000$ e $w = 24$ e $s = 1$ otteniamo $\frac{1000-24}{1} = 9976$ campioni.

Disponiamo i campioni in una matrice

$$X_{k,w} = \begin{bmatrix} [t_1 & \cdots & t_w] \\ [t_2 & \cdots & t_{w+s}] \\ & [\cdots] & \\ [t_{k \cdot s} & \cdots & t_{k \cdot s+w}] \end{bmatrix}$$

Parte III

Codice

Capitolo 4

Appunti sul codice bitcoin-core

Parte IV

Appendici

Appendice A

Processo degli arrivi di Poisson

Appendice B

Glossario

Bibliografia

[1] Eric Budish. The economic limits of bitcoin and the blockchain. 2018.

[2] Satoshi Nakamot. Bitcoin: A peer-to-peer electronic cash system. 2008.

[3] Pieter Wuille. Schnorr signatures for secp256k1. 2018.