Principles of Cryptocurrency Design

Appunti ed Esercizi

Francesco Pasquale

16 aprile 2026

I blocchi della Blockchain di Bitcoin sono delle strutture concettualmente divise in due parti: `header` e `data`. Abbiamo studiato l'`header` nelle scorse due lezioni. In questa iniziamo a vedere il contenuto della parte `data`, ma prima di tutto facciamo un po' di riscaldamento con un paio di esercizi relativi ancora all'`header`.

Esercizio 1. Prendiamo un blocco appena creato, il n. `945350`), della blockchain di Bitcoin.

1. Usando i programmi scritti nella lezione precedente e l'header del blocco, verificare che il timestamp del blocco è 1776355415, ossia lo unix time corrispondente a *gio 16 apr 2026, 18:03:35*, e che il target di quel blocco è $132740 * 2^{160}$.
2. Sapendo che il `target` è impostato in modo che, in media, viene creato un nuovo blocco ogni dieci minuti, stimare l'*hash rate* h (ossia il numero di hash per secondo) che i miner stanno complessivamente eseguendo attualmente.
3. Supponete che un extraterrestre arrivi sulla terra con un *hardware* che riesce a fare k volte più hash al secondo di quanti ne fanno tutti i miner attuali messi insieme, per qualche $k > 1$, e decida di voler usare il suo hardware per riscrivere l'intera blockchain di Bitcoin a partire dal *genesis block*. Assumendo che l'hash rate degli altri miner rimanga costante, calcolare approssimativamente quanto tempo passerebbe prima che, in accordo al protocollo Bitcoin, tutti riconoscano la blockchain dell'extraterreste come quella valida.

Esercizio 2. Assumendo che la funzione crittografica SHA256 sia un *random oracle*, ossia supponendo che dato qualunque $y \in \{0,1\}^{256}$ ogni volta che calcoliamo l'hash di un nuovo $x \in \{0,1\}^*$ si abbia che $\mathbf{P}\left(\text{SHA256}(x) = y\right) = 2^{-256}$. Qual è la probabilità che il prossimo blocco impieghi più di k minuti per essere creato, per $k > 10$?

Nel linguaggio comune, quando parliamo di un *bitcoin* tipicamente intendiamo una unità della criptovaluta corrispondente (un po' come parliamo di un *euro* o di un *dollaro*). In realtà, non esiste nessun "oggetto digitale" che corrisponda a un *bitcoin* (o a una frazione o a un multiplo di bitcoin), ciò che invece esiste come oggetto digitale nel sistema Bitcoin sono le *transazioni*: il contenuto della parte che abbiamo chiamato `data` di un blocco della Blockchain di Bitcoin è infatti una sequenza di transazioni. Per capire come l'astrazione di un *bitcoin* come una unità di criptovaluta è legata alle transazioni dobbiamo dare un'occhiata più approfondita alla struttura di queste ultime.

1 Transazioni

Nella sua versione originale[1], una *transazione* `tx`, è una struttura formata dai campi *inputs* e *outputs*, da un numero di *versione* e da un *locktime*. I campi *versione* e *locktime* hanno una dimensione

[1] La struttura delle transazioni in Bitcoin ha avuto una evoluzione nell'Agosto 2017 con quello che viene denominato upgrade SegWit, di cui parleremo in seguito.

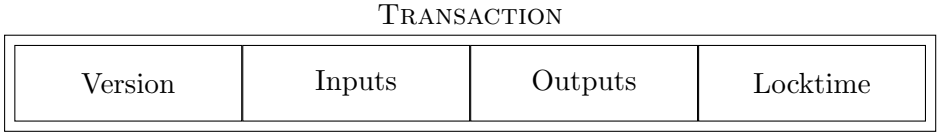

Figura 1: La struttura di una transazione nella versione originale di Bitcoin

di 4 byte ciascuno, i campi *inputs* e *outputs* sono di lunghezza variabile. Ogni transazione contiene 0 o più `input` (le uniche transazioni che possono contenere 0 input sono le cosiddette transazioni *coinbase* di cui parleremo fra un attimo, tutte le altre devono averne almeno uno) e uno o più `output`. Ogni `input` contiene un puntatore a un `output` di un'altra transazione. Diciamo che una transazione $\texttt{tx}_a$ *spende* l'output $\texttt{tx}_b\texttt{.output}$ di una precedente transazione $\texttt{tx}_b$ se uno degli input di $\texttt{tx}_a$ punta all'output $\texttt{tx}_b\texttt{.output}$. Ogni `output` può essere speso da una sola transazione, se ci sono due o più transazioni che spendono lo stesso output, una sola sarà quella considerata "valida": quella che viene inserita in un blocco. Se un blocco dovesse contenere una transazione che spende un output già speso da una transazione in un blocco precedente (oppure se il blocco dovesse contenere due o più transazioni che spendono lo stesso output - non speso - di una transazione precedente), allora il blocco non sarebbe considerato valido e non verrebbe aggiunto alla catena dai nodi onesti.

1.1 Input e output

Un `output` contiene due campi: un `valore`, ossia un numero intero che corrisponde all'ammontare di quell'output e un `locking_script`. Un `input` è una struttura formata da (1) un puntatore all'output di una precedente transazione, indicato con la coppia (`prev_tx`, `prev_id`), dove `prev_tx` è la transazione e `prev_id` è l'indice dell'output all'interno della transazione e (2) un `unlocking_script` (ogni transazione ha anche un *sequence number*, ma per il momento non ce ne occupiamo).

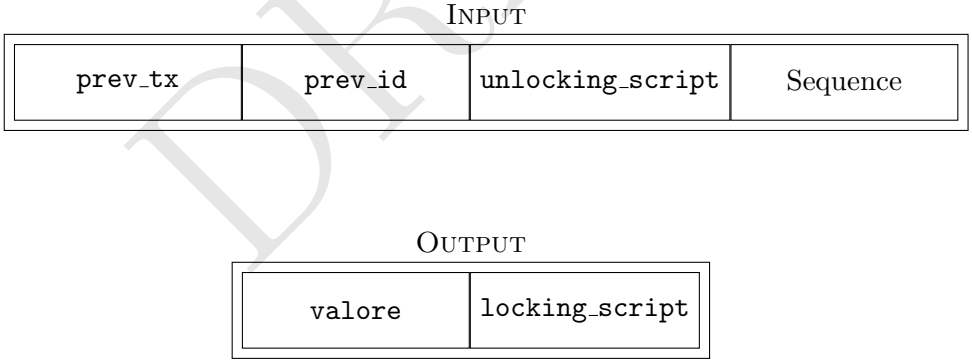

Figura 2: La struttura di un input e di un output

Discuteremo in dettaglio gli script in una delle prossime lezioni. Per il momento diciamo solo che i due script, `locking_script` nell'output e `unlocking_script` nell'input, sono dei veri e propri "programmi" scritti in un linguaggio che tutti i nodi della rete possono interpretare. Data una transazione `tx` e un suo input `tx.input`, se la concatenazione dello *unlocking script* nell'input, `tx.input.unlocking_script`, con il *locking script* contenuto nell'output della transazione precedente puntato da quell'input, `tx.input.(prev_tx, prev_id).locking_script` forma un programma che restituisce True, allora l'input `tx.input` è *valido*. Nel gergo di Bitcoin il `locking_script` viene chiamato *scriptPk* e lo `unlocking_script` viene chiamato *scriptSig* perché nella loro forma più semplice questi due script contengono una *public key* e una *signature* rispettivamente, e la concatenazione dei due restituisce True se e solo se la *signature* è una firma della transazione, valida rispetto alla *public key*. Una transazione `tx` è valida se (1) tutti i suoi input puntano a degli

output che non sono già stati spesi da qualche altra transazione e sono validi, e (2) se la somma dei `valore` degli output, $\sum$ `tx.output.valore` è minore o uguale alla somma degli `valore` degli output puntati dagli input, $\sum$ `tx.input.(prev_tx, prev_id).valore`. La differenza, maggiore o uguale a zero, fra questi due valori è la *transaction fee* della transazione `tx`.

$$\texttt{tx.fee} = \sum_{\texttt{tx.input}} \texttt{tx.input.(prev\_tx, prev\_id).valore} - \sum_{\texttt{tx.output}} \texttt{tx.output.valore} \tag{1}$$

2 Transazione *coinbase* e la "coniazione" di nuovi bitcoin

Abbiamo detto che gli input di ogni transazione devono puntare ad output di altre transazioni e che c'è un'unica transazione in ogni blocco, la prima, che non ha input: questa è la cosiddetta *coinbase transaction*, nell'output della coinbase transaction ci sono bitcoin che non esistevano prima, quindi quelli "coniati" con quel blocco.

Affinché la *coinbase transaction* dell'i-esimo blocco, per $i = 1, 2, \ldots$, sia valida, l'ammontare contenuto nell'output deve essere quello specificato dal protocollo per il blocco i-esimo, `new_btc`[i], più la somma delle *transaction fee* delle altre transazioni inserite nel blocco. L'ammontare specificato dal protocollo per i nuovi bitcoin creati per ogni blocco parte da 50 e si dimezza ogni 210000 blocchi

$$\texttt{new\_btc}[i] = \frac{50}{2^{\lfloor i/210000 \rfloor}} \tag{2}$$

in questo modo, il totale dei bitcoin in circolazione sarà sempre minore o uguale a 21 milioni.

Esercizio 3. Considerando che il valore più piccolo di bitcoin inseribili in un output è 1 *satoshi*, che corrisponde a 10^{-8} bitcoin, e tenendo conto che per generare 210000 blocchi ci vogliono all'incirca 4 anni al ritmo di uno ogni dieci minuti, calcolare a partire da che anno i nuovi blocchi non genereranno più nuova valuta, in accordo alla Formula 2.