Principles of Cryptocurrency Design

Appunti ed Esercizi

Francesco Pasquale

16 aprile 2024

Nella lezione di oggi abbiamo implementato alcune parti di un protocollo *Bitcoin-like* per generare una catena di blocchi basato su *proof-of-work*. Gli esercizi di questa nota si riferisco al codice scritto in aula, che potete scaricare qui: `https://www.mat.uniroma2.it/~pasquale/dida/aa2324/pcd/pcd240416.zip`

Esercizio 1. Scrivere un programma che legga un file in cui ogni riga è una sequenza di 152 caratteri esadecimali e verifichi se la sequenza di righe definisce una catena *valida*. Una catena è valida se:

1. Ogni riga del file corrisponde alla serializzazione di un'istanza dell'oggetto `Block`, così come definito in dal metodo `serialize` alla riga 35 in `block.py`, espressa in esadecimale.
2. Il `prev_hash` del blocco nella prima riga è l'hash del *genesis block* che si trova nel file `config`, e per ogni $i > 1$ il `prev_hash` del blocco nella riga i-esima è l'hash del blocco nella riga $(i-1)$-esima
3. Per ogni riga, l'hash del blocco in quella riga è inferiore al target.

Nella directory `testcases/` trovate due file con due catene da mille blocchi ciascuna: una delle due catena è valida, l'altro no. Per il file che contiene la catena non valida, identificare la prima riga non valida e quale dei punti qui sopra non è soddisfatto.

Esercizio 2. Nel codice scritto a lezione abbiamo usato un un `target` costante. Definire un tempo medio `DELTA` che vogliamo imporre fra la creazione di due blocchi consecutivi (diciamo `DELTA = 120` secondi) e un numero `EPOCH_LEN` che corrisponde al numero di blocchi prima di riaggiornare il target (diciamo `EPOCH_LEN` $= 60$). Inserire i due valori nel file `config`.

Modificare il codice in `block.py` in modo che, `target`(0) sia quello del *genesis block* e viene usato per i primi `EPOCH_LEN` blocchi, ma ogni volta che il numero di blocchi creati è $k *$ `EPOCH_LEN` per qualche $k \geqslant 1$, per i successivi `EPOCH_LEN` blocchi il target sia dato dalla formula

$$\texttt{target}(k) = \texttt{target(k-1)} \cdot \frac{\delta}{\texttt{DELTA}}$$

dove con δ abbiamo indicato la differenza fra il `timestamp` del blocco $k *$ `EPOCH_LEN` e il `timestamp` del blocco $(k-1) *$ `EPOCH_LEN`.

Esercizio 3. Alla linea 58 del file `block.py` incrementiamo `block.nonce` di una unità, senza preoccuparci del fatto che questo numero deve essere a 4 byte e quindi non può in ogni caso superare $2^{32} - 1$.

Modificare il codice inserendo la condizione che, se `block.nonce` dovesse raggiungere 2^{32}, `block.nonce` ripartirebbe da 0 e `block.timestamp` verrebbe aggiornato.

Esercizio 4. Scrivere un programma che legga un file e, se contiene una catena *valida* così come definito nell'Esercizio 1, restituisca la *proof-of-work* della catena, ossia la somma, per ogni blocco, di 2^{256} diviso il `target` del blocco.