Principles of Cryptocurrency Design

Appunti ed Esercizi

Francesco Pasquale

13 aprile 2026

Nella lezione di oggi abbiamo implementato alcune parti del protocollo *Bitcoin* 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/aa2526/pcd/pcd260413.zip`

Esercizio 1. Il programma `run.py` genera una sequenza di (header) di blocchi a partire dal genesis block di Bitcoin e li scrive a video.

1. Modificare il programma per fare in modo che ogni header di 80 byte venga scritto anche su un file `blockchain.dat`.
2. Scrivere un programma che legga un file `blockchain.dat` e verifichi se contiene una catena di blocchi consistente con lo schema implementato. Ossia che
   (a) I primi 80 byte coincidono con quelli del *genesis block* di Bitcoin.
   (b) Ogni blocco successivo di 80 byte contiene nel campo `prev_hash` l'hash del blocco precedente;
   (c) L'hash di ogni blocco è inferiore al `target` contenuto nel blocco.

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 (per esempio, `DELTA = 120` secondi) e un numero `EPOCH_LEN` che corrisponde al numero di blocchi prima di riaggiornare il target (per esempio, `EPOCH_LEN` $= 60$).

Modificare il codice in `block.py` in modo che ogni volta che il numero di blocchi creati è $k * \texttt{EPOCH\_LEN}$ 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 $k * \texttt{EPOCH\_LEN}$ e il `timestamp` del blocco $(k-1) * \texttt{EPOCH\_LEN}$.

Esercizio 3. Scrivere un programma che legga un file `blockchain.dat` e

1. Verifichi se contiene una catena che soddisfa i due punti dell'Esercizio 1;
2. Verifichi che ogni blocco contenga un target consistente con quello definito nell'Esercizio 2;
3. Restituisca la *proof-of-work* della catena, ossia la somma, per ogni blocco, di 2^{256} diviso `target` $+ 1$.

Esercizio 4. Scrivere un programma che legga un file `blockchain.dat` e, se contiene una catena che soddisfa i punti degli Esercizi 1 e 2, inizi ad aggiungere blocchi alla catena.