Principles of Cryptocurrency Design

Appunti ed Esercizi

Francesco Pasquale

2 marzo 2026

In questo corso studieremo le idee fondamentali che hanno portato alla realizzazione dei moderni sistemi di criptovalute. Uno dei vari aspetti critici di questi sistemi è la necessità di raggiungere un "consenso", da parte di agenti dislocati in posizioni diverse, sulla cosiddetta *blockchain*. Il problema del consenso era già stato ampiamente studiato negli anni '80 in contesti diversi. In questa prima parte del corso perciò faremo una panoramica su alcuni dei risultati fondamentali ottenuti in quegli anni, per poter apprezzare sia i risultati stessi sia il cambiamento di paradigma introdotto da Bitcoin, la prima criptovaluta che si è diffussa su larga scala.

1 Il problema Byzantine Broadcast

Si consideri il seguente sistema distribuito:

- Un insieme di n nodi, $[n] = \{1, 2, \ldots, n\}$;
- Ogni nodo può eseguire computazioni locali arbitrarie e inviare messaggi di lunghezza arbitraria a ogni altro nodo.

Il problema computazionale di cui ci occupiamo è chiamato *Byzantine Broadcast (BB)* [2]: Degli n nodi, ce ne sono f *corrotti*, con $0 \leqslant f \leqslant n$, gli altri $n - f$ nodi sono *onesti*. Al nodo 1 (la *sorgente*) viene affidato in input un messaggio b. Vogliamo progettare un *protocollo* al termine del quale ogni nodo onesto i dia in output un valore y_i tale che

- **Consistency**: Se i e j sono due nodi onesti, allora $y_i = y_j$;
- **Validity**: Se la sorgente è un nodo onesto, allora $y_i = b$ (il messaggio affidato in input alla sorgente) per ogni nodo onesto i.

Gli $n-f$ nodi onesti applicheranno il nostro protocollo. Gli f nodi corrotti non sappiamo come si comporteranno: la nostra assunzione *worst-case* è che tutti i nodi corrotti si coalizzeranno per far fallire il nostro protocollo. Non sappiamo chi sono i nodi corrotti.

2 Modelli di calcolo distribuito e assunzioni

Il modo in cui possiamo progettare e analizzare rigorosamente protocolli distribuiti per un problema come il Byzantine Broadcast dipende drasticamente dal *modello* che usiamo, ossia da ciò su cui assumiamo di poter o non poter contare. In particolare, qui ci concentreremo su queste tre questioni:

1. Synchronous / Asynchronous. *Quando un nodo invia un messaggio, possiamo assumere che il messaggio arrivi a destinazione? Se sì, possiamo assumere un upper bound al tempo che impega un messaggio per arrivare a destinazione?* A seconda che la risposta a questa domanda sia affermativa o negativa diremo che ci troviamo in un modello sincrono o asincrono. In letteratura si trovano anche "gradazioni intermedie" note come modelli *parzialmente sincroni*.

2. Authenticated / Non authenticated. *Quando un nodo "j" riceve un messaggio proveniente da un nodo "i", il nodo "j" può assumere che il mittente del messaggio sia effettivamente "i" oppure dobbiamo considerare la possibilità che qualche nodo corrotto possa far credere al nodo "j" di essere il nodo "i" quando questo non è vero? Quando un nodo "j" riceve un messaggio da un nodo "i" e lo inoltra a un nodo "k", il nodo "k" ha modo di verificare che il messaggio era stato effettivamente scritto da "i"?* Vedremo che le risposte a queste domande distinguono se ci toviamo in un modello autenticato o non autenticato. In particolare la risposta alla seconda domanda è affermativa se c'è un PKI (Public Key Infrastructure) setup.

3. Permissioned / Permissionless. *Gli agenti che partecipano al protocollo sono fissati dall'inizio? oppure nuovi nodi possono aggiungersi durante l'esecuzione del protocollo?* In tutti i risultati degli anni '80 si assume che gli n nodi sono fissati all'inizio. Oggi tipicamente chiamiamo un tale modello *permissioned*, per distinguerlo dal modello *permissionless* usato da Bitcoin e altre criptovalute, nel quale nuovi nodi possono aggiungersi in ogni momento durante l'esecuzione del protocollo.

3 Il modello sincrono, permissioned, con PKI setup

Per iniziare, mettiamoci nella configurazione più favorevole possibile. Assumiamo di essere in un sitema *sincrono*:

- C'è un *global clock* noto a tutti i nodi che scandisce il tempo in *round* discreti: $0, 1, 2, \ldots$;
- Ogni messaggio inviato in un round t arriva a destinazione prima dell'inizio del round $t + 1$.

Assumiamo che ci sia un *Public Key Infrastructure (PKI)* setup:

- Ogni nodo i ha una coppia di chiavi (sk_i, pk_i);
- L'insieme delle chiavi pubbliche $\{pk_i : i \in [n]\}$ è *common knowledge* fra i nodi;
- Dato un messaggio m, indichiamo con $\langle m \rangle_i$ il messaggio m con l'aggiunta di una firma valida eseguita con la chiave segreta sk_i.

Esercizio 1. Mostrare un protocollo banale che funziona nel caso in cui l'insieme dei nodi corrotti è noto.

Esercizio 2. Mostrare un protocollo banale che soddisfa la condizione di *validity* ma non soddisfa la condizione di *consistency*[1] e un protocollo banale che, viceversa, soddisfa *consistency* ma non *validity*.

[1] *Consistency*: Tutti i nodi onesti danno in output lo stesso valore; *Validity*: Se la sorgente è onesta e riceve in input b, tutti i nodi onesti danno in output b.

Esercizio 3. Spiegare perché nessuno dei due protocolli seguenti funziona:

Algorithm 1 Tentativo 1

ROUND 0. La sorgente (il nodo 1) riceve in input b.
ROUND 1. La sorgente invia $\langle b \rangle_1$ a tutti i nodi.
ROUND 2. Ogni nodo i:
se nel ROUND 1 ha ricevuto $\langle \hat{b} \rangle_1$, invia $\langle \hat{b} \rangle_i$ a tutti i nodi;
altrimenti invia $\langle 0 \rangle_i$ a tutti i nodi.
ROUND 3. Ogni nodo i:
se c'è un unico valore $\hat{b}$ per cui nel ROUND 2 ha ricevuto più di $n/2$ messaggi $\langle \hat{b} \rangle_j$, restituisce in output $\hat{b}$;
altrimenti restituisce in output 0.

Algorithm 2 Tentativo 2

ROUND 0. La sorgente (il nodo 1) riceve in input b;
ROUND 1. La sorgente invia $\langle b \rangle_1$ a tutti i nodi;
ROUND 2. Ogni nodo i
se nel ROUND 1 ha ricevuto $\langle \hat{b} \rangle_1$, invia $\langle \hat{b} \rangle_i$ a tutti i nodi;
altrimenti invia $\langle 0 \rangle_i$ a tutti i nodi.
ROUND 3. Ogni nodo i
se nel ROUND 2 ha ricevuto $\langle \hat{b} \rangle_j$ da ogni altro nodo j, restituisce in output $\hat{b}$;
altrimenti restituisce in output 0.

Esercizio 4. Provare a progettare un protocollo per BB nel modello sincrono con PKI: dimostrarne la correttezza in funzione del numero f di nodi corrotti, oppure trovare un attacco che gli f nodi corrotti possono mettere in atto per far fallire il protocollo.

Nella prossima lezione vedremo il protocollo di Dolev-Strong [1] che, nel modello sincrono con PKI setup, funziona qualunque sia il numero di nodi corrotti f.

Esercizio 5. Scaricate un software che implementi lo standard OpenPGP e generate una vostra coppia di chiavi. Inviatemi poi una mail a `pasquale@mat.uniroma2.it` cifrandola con la mia chiave pubblica (che si trova sulla mia pagina web) e firmandola con la vostra chiave. Nella mail indicate nome, cognome e numero di matricola e allegate la vostra chiave pubblica.

Riferimenti bibliografici

[1] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. `https://www.cs.huji.ac.il/~dolev/pubs/authenticated.pdf`.

[2] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, pages 382–401, July 1982. `https://dl.acm.org/doi/pdf/10.1145/357172.357176`.