Principles of Cryptocurrency Design

Appunti ed Esercizi

Francesco Pasquale

12 marzo 2026

Nella lezione precedente abbiamo visto che senza PKI setup nessun protocollo per *Byzantine Broadcast* può garantire contemporaneamente *validity* e *consistency* se ci sono $f \geqslant n/3$ nodi corrotti. Qui vediamo un semplice protocollo *probabilistico* che, se il numero di nodi corrotti è $f < n/3$, garantisce *validity* con probabilità 1 e soddisfa *consistency* con alta probabilità *(w.h.p.)*[1]. Il protocollo fa uso di una funzione H che nella dimostrazione assumiamo essere un *random oracle* (per approfondire la nozione di *random oracle* si veda [1]). In una implementazione pratica, al posto di un random oracle si usa una funzione hash crittografica.

1 Un protocollo probabilistico per BB

Sia $H : \mathbb{N} \cup \{0\} \to [n]$ una funzione tale che $H(0) = 1$ e per ogni $t \in \mathbb{N}$ si comporta come un random oracle. Consideriamo il protocollo seguente per il problema Byzantine Broadcast.

Algorithm 1 Synchronous Randomized Byzantine Broadcast

1: ROUND 0. Ogni nodo u inizializza una variabile $\mathtt{sb}_u = \perp$.
   La sorgente (il nodo 1) riceve in input b e inizializza $\mathtt{sb}_1 = b$.
2: PER OGNI ITERAZIONE $t = 0, \ldots, k-1$:
3: ROUND $3t$. Si seleziona il *leader* ℓ_t dell'iterazione utilizzando il random oracle $\ell_t = H(t)$
4: Se $\mathtt{sb}_{\ell_t} \neq \perp$ allora ℓ_t invia a tutti $\mathtt{sb}_{\ell_t}$;
5: Altrimenti ℓ_t sceglie un bit $\{0, 1\}$ u.a.r. e lo invia a tutti.
6: ROUND $3t + 1$. Ogni nodo u:
7: Se $\mathtt{sb}_u \neq \perp$ allora u invia a tutti $\mathtt{sb}_u$;
8: Altrimenti u invia a tutti il bit ricevuto nel round $3t$ dal leader ℓ_t (se non ha ricevuto nulla da ℓ_t o ha ricevuto entrambi i bit, invia 0 o 1 arbitrariamente)
9: ROUND $3t + 2$. Ogni nodo u:
10: Se c'è un bit $\hat{b}$ che u ha ricevuto nel round $3t + 1$ da almeno $2n/3$ nodi distinti allora u imposta $\mathtt{sb}_u = \hat{b}$;
11: Altrimenti imposta $\mathtt{sb}_u = \perp$.
12: ROUND $3k$. Ogni nodo u:
13: OUTPUT $\mathtt{sb}_u$

A lezione abbiamo dimostrato il teorema seguente.

Teorema 1. Se il numero di nodi corrotti è $f < n/3$ allora il protocollo in Algorithm 1 soddisfa *Validity* con probabilità 1 e *Consistency* con probabilità almeno $1 - (2/3)^k$.

[1] Data una successione di eventi $\{\mathcal{E}_n\}_n$ diciamo che $\mathcal{E}_n$ avviene *con alta probabilità* (e scriveremo *w.h.p.* per *with high probability*) se esistono un $n_0 \in \mathbb{N}$ e un $c > 0$ tali che $\mathbf{P}(\mathcal{E}_n) \geqslant 1 - n^{-c}$ per ogni $n \geqslant n_0$. Nel caso in questione l'evento $\mathcal{E}_n$ è "In una sistema con n nodi il protocollo soddisfa consistency"

Sketch of Proof. Per *Validty* osservare che, se la sorgente è onesta, nell'iterazione $t = 0$ succede questo: nel ROUND 0 tutti i nodi onesti ricevono dalla sorgente il bit b (linea 4); nel ROUND 1 ogni nodo onesto invia b a tutti (linea 8); nel ROUND 2 ogni nodo onesto u riceve b da almeno $2n/3$ nodi distinti e quindi fissa il suo $\texttt{sb}_u = b$. Nelle iterazioni successive $t = 1, \ldots, k-1$ ogni nodo onesto u continua a inviare $\texttt{sb}_u = b$ (linea 7) e quindi il valore di $\texttt{sb}_u$ rimane b (linea 10). Per *Consistency* la dimostrazione procede come segue:

1. Si dimostra che in ogni iterazione t non possono esserci due nodi onesti u e v tali che alla fine dell'iterazione t abbiano $\texttt{sb}_u = 0$ e $\texttt{sb}_v = 1$;
2. Si definisce una iterazione t *lucky* se si verifica che (i) il leader ℓ_t è onesto e il bit inviato dal leader (linee 4-5) è lo stesso posseduto da ogni altro nodo onesto u che abbia $\texttt{sb}_u \neq \perp$ all'inizio dell'iterazione t;
3. Si osserva che, se esiste una iterazione *lucky* allora tutti i nodi onesti danno in output lo stesso bit;
4. Si simostra che, qualunque cosa sia successa nelle iterazioni precedenti, una iterazione t è *lucky* con probabilità almeno $1/3$;
5. Si osservare che la probabilità che fra le k iterazioni nemmeno una sia *lucky* quindi è minore di $(2/3)^k$.

□

Esercizio 1. Completare i dettagli mancanti nello sketch di dimostrazione del Teorema 1.

Esercizio 2. Si consideri il protocollo in Algorithm 1. Dato $\delta > 0$, quanto deve essere grande il numero di iterazioni $k = k(\delta)$ affinché la probabilità che l'algoritmo soddifi la condizione di *consistency* sia almeno $1 - \delta$? Quanto deve essere grande k se scegliamo $\delta = 1/n$? Confrontare il numero di round di questo protocollo con quello del protocollo di Dolev-Strong.

Esercizio 3. Alle linee 10-11 il protocollo stabilisce che ogni nodo u deve decidere come impostare il bit $\texttt{sb}_u$ a seconda che abbia ricevuto o no almeno $2n/3$ "voti" per uno specifico bit $\hat{b}$ nel round precedente.

1. Mostrare un attacco che i nodi corrotti potrebbero mettere in atto se la decisione fosse presa in funzione di un numero di voti $h < 2n/3$;

2. Mostrare un attacco che i nodi corrotti potrebbero mettere in atto se la decisione fosse presa in funzione di un numero di voti $h > 2n/3$.

Esercizio 4. Cosa succederebbe se non imponessimo che $H(0) = 1$ (ossia che il leader ℓ_0 dell'iterazione $t = 0$ è il nodo sorgente)?

Riferimenti bibliografici

[1] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993. `https://dl.acm.org/doi/pdf/10.1145/168588.168596`.

A Bonus track: Il test di primalità di Fermat

Esercizio 5. Scrivere un programma per verificare che il valore di n della chiave pubblica RSA dell'Esercizio 7 negli appunti del 5 marzo è il prodotto dei due numeri seguenti

`p = 0xbd6cc38192098eccbe0b11bfe35b8b12e028262daa151d8a47695b32cda6d613209efd3c617d6226de07d0f77d1b228b945e90dd67973754d0a47973d0001677`

`q = 0xfacaac1a37b9a3d8438a60d167cb8f56f62a7b6c0c3cc397f067dbff1ae032ab090cdcfbe47860472ca666ac525dfa275eee5ff862ce531f3765f9b321966e63`

e calcolare il d della corrispondente chiave segreta.

Esercizio 6. Dimostrare il seguente Teorema di Fermat

> **Teorema 2** (Fermat, 1640)**.** Se $p \in \mathbb{N}$ è un numero primo, allora $a^{p-1} \equiv 1 \bmod p$ per ogni $a \in \{1, \dots, p-1\}$.

(Hint: Sia $S = \{1, 2, \dots, p-1\}$ e $aS = \{a \bmod p, 2a \bmod p, \dots, (p-1)a \bmod p\}$. Osservare che siccome p è primo si ha che $S = aS$ e quindi anche $\prod_{x \in S} x \bmod p = \prod_{x \in aS} x \bmod p$. Quanto valgono questi due prodotti?)

Esercizio 7. Si consideri il seguente *test di primalità* basato sul Teorema 2

Algorithm 2 Fermat primality test

```
1. from random import randint
2.
3. def Fermat_test(n, t = 10):
4.     for _ in range(t):
5.         a = randint(2, n-2)
6.         if pow(a, n-1, n) != 1:
7.             return False
8.     return True
```

1. Osservare che se l'algoritmo restituisce False, per il Teorema 2 siamo sicuri che il numero in input non è primo, mentre se l'algoritmo restituisce True possiamo solo dire che il numero n in input ha *passato il test di Fermat*;
2. Sia $n \in \mathbb{N}$ e indichiamo con W_n l'insieme
$$W_n = \left\{a \in \{1, \dots, n-1\} \,:\, a^{n-1} \not\equiv 1 \bmod n\right\}.$$
Osservare che se n è *composto* (ossia, non è primo), allora la probabilità che n passi il test di Fermat è $(1 - |W_n|/n)^t$.
3. Sia n composto. Dimostrare che se esiste un $a \in \{1, \dots, n-1\}$ tale che $\gcd(a, n) = 1$ e $a^{n-1} \not\equiv 1 \bmod n$ allora $|W_n| \geqslant n/2$.

   (Hint: Sia a tale che $\gcd(a, n) = 1$ e $a^{n-1} \not\equiv 1 \bmod n$. Osservare che se $b \in [n] \setminus W_n$ allora $ab \bmod n \in W_n$ e che la funzione $f : [n] \setminus W_n \to W_n$, che associa a ogni $b \in [n] \setminus W_n$, $f(b) = ab \bmod n$ è iniettiva)

Esercizio 8. Un numero di Carmichael è un numero composto n, tale che $a^{n-1} \equiv 1$ per tutti gli $a \in \{1, \dots, n-1\}$ tali che $\gcd(a, n) = 1$. Per i numeri di Carmichael perciò non possiamo dire che $|W_n| \geqslant n/2$.

- Scrivere un programma che prenda in input n e calcoli $|W_n|$;
- Calcolare $|W_n|$ per tutti i numeri minori di 10000, notando in particolare il valore di $|W_n|$ quando n è uno dei primi sette numeri di Carmichael: `561, 1105, 1729, 2465, 2821, 6601, 8911`.