# Principles of Cryptocurrency Design

Appunti ed Esercizi

Francesco Pasquale

5 marzo 2024

## 1 Byzantine Broadcast nel modello sincrono, permissioned, con PKI setup

Si consideri il seguente sistema distribuito:

- C'è 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.

E il problema *Byzantine Broadcast (BB)*: 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 $y_i = b$ 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.
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 noto a tutti a priori;
- 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 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 2.** Mostrare un protocollo banale nel caso in cui l'insieme dei nodi corrotti è noto.

**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.

Nella prossima lezione vedremo il protocollo di Dolev-Strong [1]. Cercate di risolvere il prossimo esercizio in autonomia prima di vedere la soluzione trovata da Dolev e Strong.

**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.

## 2 Digital signatures

Nella prossima lezione faremo un richiamo sugli schemi di firme digitali. Per il momento, provate a risolvere in autonomia il seguente esercizio pratico. Se necessario cercate di recuperare in rete il background di cui avete bisogno.

**Esercizio 5.** Il codice Python seguente[2]

```
from Crypto.PublicKey import RSA
from hashlib import sha256

keyPair = RSA.generate(bits = 1024)

msg = b'Benvenuti al corso di Principles of Cryptocurrency Design - AA 23/24!'
msg_hash = int.from_bytes(sha256(msg).digest(), byteorder = 'big')
msg_sig = pow(msg_hash, keyPair.d, keyPair.n)

print("Firma: ", hex(msg_sig))
```

[2]La libreria `hashlib` è built-in Python. Per `Crypto` usare la libreria `pycryptodome` https://pypi.org/project/pycryptodome/

ha scritto a video questa stringa:

```
Firma: 0x51b145b9413e2e80f58093391baf293a6b61adb4b572ec46aca3ba67c59b1d4ce33c
95ac0c77458515a681ff46d2b52bd6113fae4bd8e85123cc85cd742db9b9b83b9cf1d93b59e9b9
0f6d96b1e137a9c0a8e02ebd79ecd3ea755c6c95df62bbd37a9926646fd19f1ac1830b9c8eb9de
f9c4aa2704c4b087963956e3815a306e
```

Scrivere un programma per verificare che si tratta di una firma valida del messaggio *Benvenuti al corso di Principles of Cryptocurrency Design - AA 23/24!* generata con la chiave pubblica

```
n = 0x9cf5e1b3547f43927df163fe6ee1fd6896f458a2bc1024f9f0b926df34e66bc8820c30cf
c88d26c67192f7f10b193f0dd61ddf6b9a01ccbbee12be18de25c62faf71bc71fd1822213a995f
3cac2f74496fa11c87d24826ac311dff0264519fe00d93f5d15718fe6c6aedd520eea29f46fe15
815cb73074b744951dd5092b8407
```

```
e = 0x10001
```

## Riferimenti bibliografici


[1] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.