# Principles of Cryptocurrency Design

Appunti ed Esercizi

Francesco Pasquale

23 aprile 2026

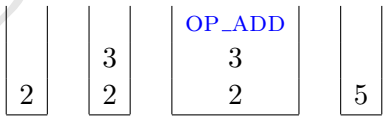
Abbiamo detto precedentemente che una transazione $\texttt{tx}_a$ può avere uno o più output $\texttt{tx}_a\texttt{.output}$, ogni output è costituito da un `valore` e da un `locking_script`. Una transazione $\texttt{tx}_b$ che volesse *spendere* $\texttt{tx}_a\texttt{.output}$ deve contenere, nell'input che punta a $\texttt{tx}_a\texttt{.output}$, un `unlocking_script` tale che la concatenazione di $\texttt{tx}_b\texttt{.input.unlocking\_script}$ con $\texttt{tx}_a\texttt{.output.locking\_script}$ formi uno *script* che restituisca TRUE. Vediamo più in dettaglio questo processo.

## 1 Script

Il software del sistema Bitcoin, in esecuzione su ogni nodo della rete, contiene l'iterprete di un semplice linguaggio di programmazione che viene chiamato Script. Il linguaggio Script è *stateless* (non ci sono variabili che possono salvare localmente un dato e andare a rileggerlo successivamente). Un programma (uno *script*) è una sequenza di comandi (*istruzioni* o *dati*) che vengono eseguiti secondo un modello *stack-based*: i *dati* e le *istruzioni* vengono inseriti in una *pila* (lo *stack*); quando una istruzione viene inserita nella pila, può estrarre uno o più dati dalla pila, eseguire delle computazioni sui dati ed eventualmente inserire il risultato delle computazioni nella pila. Facciamo un esempio. L'istruzione OP_ADD nel linguaggio Script estrae due dati dalla pila e ne esegue la somma, quindi il seguente *script*

<2> <3> OP_ADD

verrebbe eseguito dall'interprete Script presente sui nodi della rete Bitcoin in questo modo: viene inserito nella pila il dato 2, poi viene inserito nella pila il dato 3, poi viene inserito nella pila l'istruzione OP_ADD che estrae dalla pila i due dati sottostanti e inserisce nella pila 5




### 1.1 Pay-to-Public-Key (p2pk)

Il più semplice `locking_script` usato in Bitcoin è chiamato *Pay-to-Public-Key (p2pk)* ed è formato da due comandi:

<pk> OP_CHECKSIG (1)

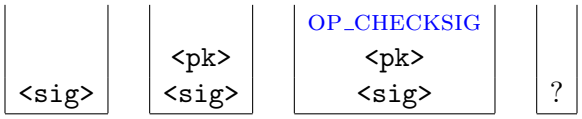
Il primo comando, <pk>, è un dato costituito da una chiave pubblica. Il secondo, OP_CHECKSIG, è una istruzione (OP_CODE nella terminologia di Bitcoin) che estrae dalla pila i due comandi sottostanti e verifica se il secondo è una firma corretta della transazione rispetto alla chiave pubblica contenuta nel primo dato, in caso affermativo inserisce nella pila TRUE e la transazione risulta valida, in caso negativo inserisce nella pila FALSE e la transazione viene rigettata come non valida. Se una transazione $\texttt{tx}_a$ ha un output $\texttt{tx}_a\texttt{.output}$ con un `locking_script` di questo tipo, allora per spendere quell'output una transazione $\texttt{tx}_b$ dovrà avere un input con un `unlocking_script` costituito da un solo comando:

<sig>

in modo che la concatenazione di $\texttt{tx}_b$.input.unlocking_script con $\texttt{tx}_a$.output.locking_script sia

<sig> <pk> OP_CHECKSIG (2)

e l'interprete Script che esegue lo script avrà sullo stack alla fine dell'esecuzione TRUE o FALSE a seconda che <sig> sia oppure no una firma della transazione $\texttt{tx}_b$ valida rispetto alla chiave pubblica <pk>




Si noti che <sig> non può contemporaneamente essere parte della transazione $\texttt{tx}_b$ ed essere una firma valida per $\texttt{tx}_b$. Infatti, più precisamente possiamo dire che, affinché lo script in (2) inserito in una transazione $\texttt{tx}_b$ ritorni TRUE, <sig> deve essere una firma valida della sequenza di byte che si ottiene da $\texttt{tx}_b$ dove nel campo di $\texttt{tx}_b$.input.unlocking_script riservato a <sig> viene inserito <pk>.

## 1.2 Pay-to-Public-Key-Hash (p2pkh)

Tutte le prime transazioni di Bitcoin avevano negli output dei locking_script del tipo *p2pk*. Una volta inviata una transazione contenente uno script di quel tipo, la chiave pubblica quindi diventa nota a tutti, e i bitcoin associati a quella chiave pubblica sono protetti "soltanto" dallo schema di firma digitale ECDSA (*Elliptic Curve Digital Signature Algorithm*, parleremo di questo schema in una delle prossime lezioni): se qualcuno dovesse riuscire a trovare un metodo per risalire a una chiave privata partendo da una chiave pubblica potrebbe spendere i bitcoin di quell'output.

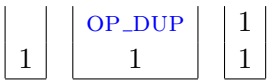
Per fornire un ulteriore livello di protezione, sin dalle origini il sistema Bitcoin prevede locking_script cosiddetti *Pay-to-Public-Key-Hash (p2pkh)* (la prima transazione con un output di questo tipo si trova nel blocco 728). In questo tipo di locking_script non compare direttamente la chiave pubblica, ma l'hash della chiave pubblica <pkhash>. Lo unlocking_script necessario per spendere un tale output deve contenere sia la chiave pubblica il cui hash corrisponde a quello nel locking_script, sia una firma valida rispetto a quella chiave pubblica. In questo modo la chiave pubblica non viene resa nota a tutti nel momento in cui la transazione con il locking_script viene inviata alla rete, ma soltanto quando quell'output viene speso.

Un locking_script del tipo *p2pkh* è formato da cinque comandi:

OP_DUP OP_HASH160 <pkhash> OP_EQUALVERIFY OP_CHECKSIG (3)

dove

- OP_DUP è un'istruzione che inserisce nello stack una copia dell'elemento sottostante. Per esempio, lo script <1> OP_DUP verrebbe eseguito in questo modo




- OP_HASH160 è un'istruzione che estrae dallo stack l'elemento sottostante e inserisce nello stack il suo hash (opportunamente calcolato);
- <pkhash> è l'hash di una chiave pubblica;
- OP_EQUALVERIFY è un'istruzione che estrae due elementi sottostanti dallo stack e se sono uguali non fa nulla, altrimenti interrompe l'esecuzione dello script dichiarando la transazione non valida;
- OP_CHECKSIG è il comando che abbiamo visto in precedenza in *p2pk*.

**Esercizio 1.** Qual è lo unlocking_script che consente di sbloccare un locking_script del tipo *p2pkh*?

**Esercizio 2.** L'istruzione OP_EQUAL estrae due elementi sottostanti dallo stack e inserisce nello stack TRUE se i due elementi sono uguali e FALSE altrimenti. Scrivere un `locking_script` che può essere sbloccato solo da chi conosce un messaggio `msg` il cui hash è `<msg_hash>`.

**Esercizio 3.** L'istruzione OP_NOT estrae un elemento dallo stack e, se è TRUE o FALSE ne inserisce la negazione nello stack, altrimenti inserisce nello stack FALSE. L'istruzione OP_VERIFY estrae un elemento dallo stack e, se è TRUE non fa nulla altrimenti interrompe l'esecuzione e la transazione è non valida. L'istruzione OP_SWAP estrae due elementi dallo stack e li reinserisce nello stack in ordine inverso. Usando queste e le altre istruzioni viste in precedenza, progettare un `locking_script` che può essere sbloccato solo con un `unlocking_script` che contenga un messaggio `<msg2>` diverso da un dato messaggio `<msg>` che però abbia lo stesso valore di hash (calcolato con OP_HASH160).

### 1.3 Pay-to-Multisig (p2m)

Ci sono situazioni reali nelle quali è necessario che la possibilità di spendere un determinato valore sia vincolata all'accordo di due o più persone (per esempio, una casa acquistata congiuntamente o ricevuta in eredità da due o più persone può essere rivenduta solo se c'è il consenso di tutti i proprietari). Nei `locking_script` del tipo *p2pk* e *p2pkh* chiunque conosca la chiave segreta associata alla chiave pubblica con cui è stato creato il `locking_script` può spendere quell'output. Questo problema potrebbe essere risolto con uno dei sistemi che si trovano in letteratura, per generare in modo congiunto una chiave segreta fra $n$ partecipanti che sia utilizzabile solo se un sottoinsieme di $t$ partecipanti collaborano (si veda, per esempio, [1]). In Bitcoin c'è un `locking_script` dedicato a questo, che chiamato *Pay-to-Multisig (p2m)*, nel quale si può inserire il numero $n$ di partecipanti, le $n$ chiavi pubbliche e la *threshold* $t$, e per spendere l'output è necessario inviare un `unlocking_script` che contenga $t$ firme valide rispetto a $t$ delle $n$ chiavi pubbliche contenute nel `locking_script`.

Per esempio, un `locking_script` *p2m* che richieda 2 firme valide su 3 per essere speso sarà fatto così

OP_2 `<pkA>` `<pkB>` `<pkC>` OP_3 OP_CHECKMULTISIG (4)

Lo `unlocking_script` corrispondente dovrà contenere due firme della transazione, valide rispetto a due delle tre chiavi pubbliche inserite nel `locking_script`, per esempio `<sigB>` `<sigC>`.

**Esercizio 4.** Provare a pensare a come potrebbe essere fatto un `locking_script` come quello in (4) che però invece delle chiavi pubbliche contenga soltanto gli hash delle chiavi pubbliche - come nei `locking_script` *p2pkh* in (3) - e a come dovrebbe essere fatto lo `unlocking_script` corrispondente necessario a spendere l'output.

### 1.4 Pay-to-Script-Hash (p2sh)

Nei `locking_script` di tipo *p2m* descritti qui sopra è chi crea l'output (il *mittente* dei fondi) che deve includere nella transazione le condizioni con cui quell'output possa essere successivamente speso. Per poter affidare questo onere al *destinatario* dei fondi invece che al mittente, sono stati successivamente introdotti gli script *Pay-to-Script-Hash (p2sh)*. Con *p2sh* è il destinatario dell'output che crea lo script con le condizioni di spesa (chiamato *redeem script*) e invia al mittente soltanto l'hash di questo script; il mittente include quindi questo hash nel `locking_script` dell'output della transazione senza sapere quale sia il *redeem script* che corrisponde a quell'hash. Quando il destinatario vuole spendere l'output deve includere nello `unlocking_script` il *redeem script*.

Un `locking_script` del tipo *p2sh* è formato semplicemente da tre comandi:

OP_HASH160 `<redeem_script_hash>` OP_EQUAL (5)

Lo `unlocking_script` corrispondente deve includere il *redeem script* il cui hash è `<redeem_script_hash>` e le corrispondenti condizioni che rendono TRUE quello script. Quando un nodo Bitcoin riceve una transazione con un input che spende un output *p2sh*, il nodo esegue prima un controllo sull'hash del redeem script per verificare che corrisponda a quello specificato nell'output; successivamente esegue il redeem script stesso per validare le condizioni di spesa.

I `locking_script` *p2sh* non esistevano nella versione originale di Bitcoin. Sono stati introdotti come *upgrade* nel 2012 a seguito della *Bitcoin Improvement Proposal* BIP-016.

### 1.5 Dati nella Blockchain (`op_return`)

Siccome la Blockchain di Bitcoin è un registro permanente e immutabile replicato su decine di migliaia di nodi, gli utenti hanno cominciato ad usarlo per "salvare" dei dati, per esempio includendo nei `locking_script` delle sequenze di byte che non corrispondevano a delle chiavi pubbliche, ma codificavano dei dati

L'istruzione OP_RETURN è stata introdotta per consentire l'inserimento di dati arbitrari nella blockchain in modo esplicito, senza creare output che formalmente risulterebbero spendibili anche se di fatto non lo sono, andando a "sporcare" lo *UTXO set*. Dal punto di vista operativo, uno script che contiene OP_RETURN termina immediatamente la propria esecuzione restituendo un fallimento nel momento in cui l'istruzione OP_RETURN viene eseguita. Tipicamente, un `locking_script` di questo tipo ha la forma OP_RETURN `<data>`, dove `<data>` è una sequenza di byte che viene semplicemente inserita nella transazione. Poiché lo script fallisce sempre quando eseguito, non esiste uno `unlocking_script` che possa sbloccare tale output per cui l'output non viene mantenuto dai nodi nello *UTXO set*.

## Riferimenti bibliografici

[1] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 522–526. Springer, 1991.