Principles of Cryptocurrency Design

Appunti ed Esercizi

Francesco Pasquale

18 maggio 2026

Nella nota precedente abbiamo visto come sia possibile in Bitcoin implementare dei *canali di pagamento*: due agenti, Alice e Bob, possono collaborativamente inserire nella Blockchain una transazione, la *funding transaction*, con un output spendibile solo da una successiva transazione che sia firmata da entrambi, e allo stesso tempo entrambi gli agenti possono mantenere *off-chain* delle transazioni di *committment* con le quali potrebbero spendere unilateralmente l'output della *funding transaction*, se l'altro agente dovesse smettere di collaborare. Usando gli opportuni protocolli crittografici, abbiamo visto come sia possibile costruire nuove *committment transactions* e "revocare" le precedenti, in modo che il *bilancio* del canale (quanti *btc* dell'output della funding transaction sono di Alice e quanti sono di Bob) possa essere aggiornato un numero arbitrario di volte, senza inviare nuove transazioni alla rete Bitcoin, dove l'unico limite al valore dei pagamenti è dato dall'ammontare complessivo di *btc* contenuto nell'output della *funding transaction*.

In questa nota vediamo come sia possibile implementare un sistema di *routing* di pagamenti all'interno di una "rete di canali": se Alice e Bob hanno un canale di pagamento e Bob e Carol hanno un canale di pagamento, allora Alice può chiedere a Bob di fare da intermediario per il suo pagamento a Carol: Alice pagherà Bob (vale a dire, ci sarà un aggiornamento del bilancio del canale fra Alice e Bob) e Bob pagherà Carol (vale a dire, ci sarà un aggiornamento del canale fra Bob e Carol).

1 Hash Time-Locked Contracts

In prima approssimazione, il meccanismo che nella Lightning Network garantisce che il *routing* del pagamento - da Alice a Carol passando per Bob come intermediario - sia eseguito in modo *trustless* è il seguente:

1. Carol genera un valore qualunque r, ne calcola l'hash $h = H(r)$ e dà h ad Alice;
2. Alice construisce una transazione con un output che può essere speso da Bob, se Bob esibisce una preimmagine di h;
3. Bob a sua volta costruisce una trasazione con un output che può essere speso da Carol, se Carol esibisce una preimmagine di h.

Siccome Carol conosce una preimmagine r di h, può inviarla a Bob e chiedergli di aggiornare il bilancio del loro canale. Se Bob non dovesse collaborare, Carol potrebbe spendere il suo output inviandolo sulla Blockchain. A sua volta Bob, una volta venuto a conoscenza della preimmagine r di h (o perché l'ha ricevuta da Carol, o perché Carol ha speso il suo output inviandolo sulla Blockchain e quindi rivelando r a tutti) può inviare r ad Alice e chiederle di aggiornare il bilancio del loro canale. Se Alice non dovesse collaborare, Bob spenderebbe il suo output e otterrebbe comunque il pagamento. Questo conclude il pagamento da Alice a Carol.

Esercizio 1. Si noti che il pagamento da Alice a Carol, anche se coinvolge due aggiornamenti di bilancio, è *atomico*: o entrambi gli aggiornamenti dei bilanci dei due canali possono andare a buon fine, oppure nessuno dei due può andarci.

Esercizio 2. Usando gli *opcodes* opportuni, progettare un `locking_script` che implementi la condizione per cui il corrispondente `unlocking_script` sia una firma valida relativa a una chiave pubblica `pk` e la preimmagine (diciamo rispetto a OP_HASH160) di un certo h.

Il meccanismo descritto nei punti 1,2,3 all'inizio di questa sezione ha il problema che, se per qualunque ragione Carol non dovesse rivelare r, Alice e Bob si troverebbero in una situazione di stallo in cui non potrebbero tornare in possesso dell'ammontare che hanno bloccato nell'output della transazione emessa. Per questo motivo, nei punti 2 e 3 Alice e Bob inseriscono un *time-lock*, ossia una condizione alternativa che gli consente di tornare in possesso dell'output nel momento in cui il numero di blocchi nella Blockchain raggiunge un valore prefissato x. Il tempo che intercorre fra il blocco corrente e il blocco x è quanto Alice e Bob dovranno aspettare prima di poter tornare in possesso in modo sicuro di quano avevano bloccato per il pagamento a Carol, se Carol dovesse non rivelare mai la preimmagine r.

1. Carol genera un valore qualunque r, ne calcola l'hash $h = H(r)$ e dà h ad Alice;
2. Alice construisce una transazione con un output che può essere speso da
   - Bob, se Bob esibisce una preimmagine di h;
   - Alice dopo un `expiry_time`.
3. Bob a sua volta costruisce una trasazione con un output che può essere speso da
   - Carol, se Carol esibisce una preimmagine di h;
   - Bob dopo un tempo `expiry_time` $- \Delta$.

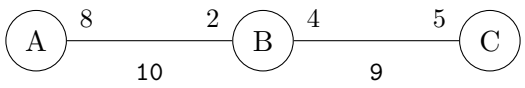
Si noti che il meccanismo descritto informalmente qui sopra deve essere implmentato con cautela, per garantire che ogni parte (Alice, Bob, o Carol) in ogni momento sia "tutelata" (possegga una transazione firmata dall'altra parte del canale che le consenta, se inviata alla rete Bitcoin, di riprendere ciò che le spetta). Vediamolo più in dettaglio con un esempio.

Supponiamo che Alice e Bob abbiano un canale di capacità 10 e Bob e Carol abbiano un canale di capacità 9. Questo significa che sulla blockchain ci sono le *funding transaction* dei due canali

FundingTx A-B	
INPUT:	`b8ff...11f0`
OUTPUT:	10*btc*, spendibili da `pkA` & `pkB`

FundingTx B-C	
INPUT:	`75a2...d41a`
OUTPUT:	9*btc*, spendibili da `pkB` & `pkC`

Supponiamo che il bilancio attuale del canale fra Alice e Bob sia di 8 per Alice e 2 per Bob; mentre il bilancio attuale del canale fra Bob e Carol sia di 4 per Bob e 5 per Carol.



Questo significa che ci sono quattro transazioni *off-chain*. Per il canale fra Alice e Bob ci sono le transazioni

CommitTx-i - Alice	
INPUT:	*FundingTx A-B*
OUTPUT 1:	8*btc*, spendibili da `pkA`, dopo n blocchi oppure da `pkA`r*i* & `pkB`, subito
OUTPUT 2:	2*btc*, spendibili da `pkB`

CommitTx-i - Bob	
INPUT:	*FundingTx A-B*
OUTPUT 1:	8*btc*, spendibili da `pkA`
OUTPUT 2:	2*btc*, spendibili da `pkB`, dopo n blocchi oppure da `pkA` & `pkB`r*i*, subito

La *CommitTx-i - Alice* è nelle mani di Alice insieme alla firma della transazione fatta da Bob con la secret key associata a pk$B$; La *CommitTx-i - Bob* è nelle mani di Bob insieme alla firma della transazione fatta da Alice con la secret key associata a pk$A$.

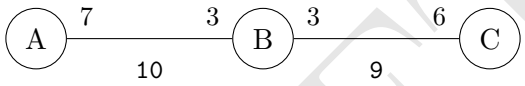
Per il canale fra Bob e Carol ci sono le transazioni

CommitTx-j - Bob	
INPUT:	*FundingTx B-C*
OUTPUT 1:	4*btc*, spendibili da pkB, dopo n blocchi oppure da pkBrj & pkC, subito
OUTPUT 2:	5*btc*, spendibili da pkC

CommitTx-j - Carol	
INPUT:	*FundingTx B-C*
OUTPUT 1:	4*btc*, spendibili da pkB
OUTPUT 2:	5*btc*, spendibili da pkC, dopo n blocchi oppure da pkB & pkCrj, subito

La *CommitTx-j - Bob* è nelle mani di Bob insieme alla firma della transazione fatta da Carol con la secret key associata a pk$C$; La *CommitTx-j - Carol* è nelle mani di Carol insieme alla firma della transazione fatta da Bob con la secret key associata a pk$B$.

Ora supponiamo che Alice voglia fare un pagamento di 1btc a Carol, sfruttando il cammino che passa per Bob. I bilanci dei due canali alla fine del pagamento dovranno quindi essere



Carol genera un "segreto" r, ne calcola l'hash $h = Hash(r)$ e lo invia ad Alice.
Alice costruice la transazione seguente e chiede a Bob di firmarla

HTLCTx - A-B - Alice	
INPUT:	*FundingTx A-B*
OUTPUT 1:	7*btc*, spendibili da pkA, dopo n blocchi oppure da pkB & pkArα
OUTPUT 2:	2*btc*, spendibili da pkB
OUTPUT 3:	1*btc*, spendibile da pkB & pkArα oppure da pkB & preimmagine di h oppure da pkA & pkB dopo `expiry_time`

L'output 3 quindi è di Bob se conosce la revocation key oppure una preimmagine di h, altrimenti è chiuso in un 2-of-2 multisig, può essere speso solo dopo il `expiry_time` e Alice deve poterlo riprendere indietro. Quindi Alice costruisce anche un'altra transazione, la *HTLC-timeout* transaction che spende l'output 3, e chiede a Bob di firmarla.

HTLC-timeoutTx - A-B	
INPUT:	*HTLCTx - A-B - Alice,* OUTPUT 3
OUTPUT 1:	1*btc*, spendibili da pkA, dopo n blocchi oppure da pkB & pkArα

Da parte sua Bob costruice la transazione seguente e chiede ad Alice di firmarla

HTLCTx - A-B - Bob	
INPUT:	*FundingTx A-B*
OUTPUT 1:	$7btc$, spendibili da `pk`A
OUTPUT 2:	$2btc$, spendibili da `pk`B dopo n bocchi oppure da `pk`A & `pk`Brα
OUTPUT 3:	$1btc$, spendibile da `pk`A & `pk`Brα oppure da `pk`A dopo `expiry_time` oppure da `pk`A & `pk`B

L'output 3 quindi è di Alice se conosce la revocation key oppure dopo il `expiry_time`, altrimenti è chiuso in uni 2-of-2 multisig e Bob deve poterlo spendere se conosce una preimmagine di h. Quindi Bob cosruisce anche un'altra transazione, la *HTLC-success* transaction che spende l'output 3, e chiede ad Alice di firmarla.

HTLC-successTx - A-B	
INPUT:	*HTLCTx - A-B - Bob,* OUTPUT 3
OUTPUT 1:	$1btc$, spendibile da `pk`B & preimmagine di h, dopo n blocchi oppure da `pk`A & `pk`Brα

Esercizio 3. Costruire le HTLC Tx corrispondenti per il canale fra Bob e Carol.

Si noti che prima dell'`expiry_time` tutte queste transazioni devono essere "revocate" (o inviate alla rete Bitcoin), altrimenti la parte ricevente non sarebbe più sicura di poter spendere l'output, anche conoscendo la preimmagine di h.

Esercizio 4. Riflettere sul perché se *`expiry_time`* di Alice è x, quello di Bob deve essere di inferiore $x - \Delta$.

Esercizio 5. Utilizzando gli OPCODE opportuni, cercare di costruire dei `locking_script` che implementino le condizioni di spesa delle transazioni. Poi guardare le specifiche in BOLT3