INFO GENERICHE

- Algorand da reward agli utenti che partecipano alla governance se dimostrano di avere una visione a lungo termine (detenendo algo per più di un tot. tempo) e partecipano alle votazioni emesse dalla AERP.
-

ALGORAND VS ETHEREUM

- Algorand Standard Assets (ASA): framework di tokenizzazione di diversi strumenti (stablecoin, nft, documenti, fidelity token ecc) in cui hanno la stessa efficienza di scalabilità, sicurezza, decentralizzazione della moneta nativa Algo. Ad esempio, per il costo di transazione, in Ethereum i token sono in smart-contract in cui viene trasferito in maniera diversa di come vengono trasferiti gli Ether. Quindi i costi di trasferimento sono più elevati rispetto ad Algorand.
- Normalmente in altre BC gli smart-contract sono in layer-2, mentre in Algorand la logica degli Smart-Contract è all'interno del layer-1 (ASC1 – Algorand Smart Contract layer-1) quindi sono situati direttamente dentro al registro. In questo modo si evita la congestione nel momento in cui non dobbiamo fare un trasferimento da layer-2 a layer-1, e aumenta il livello di sicurezza perché non c'è separazione tra layer-1 e layer-2, quindi non c'è l'impedimento del commit dello smart contract. Abbiamo una rappresentazione di asset con Algorand Standard Asset come primitiva in layer-1 e avere delle transazioni atomiche senza avere altre logiche come quella dell'Hash time lock che hanno dei limiti a livello di sicurezza.
- Svantaggio stake
- Su algorand non ci sono standard diversi ERC come su ethereum, non si verificano problemi di interoperabilità. Inter-operabilità: garantita perché tutte le primitive sono definite a layer-1 e non ha bisogno di far parlare standard diversi (ad esempio ethereum ha i vari ERC)
- Al contrario, i contratti on-chain sono molto leggeri da eseguire perché sono allo stesso livello della stessa moneta ALGO, quindi i nodi possono eseguirli quasi in tempo reale.
- Algorand Co-Chains , che sono catene private autorizzate che interagiscono con Algorand MainNet.

Una semplice panoramica di ciò che offrono le Co-Chain Algorand:

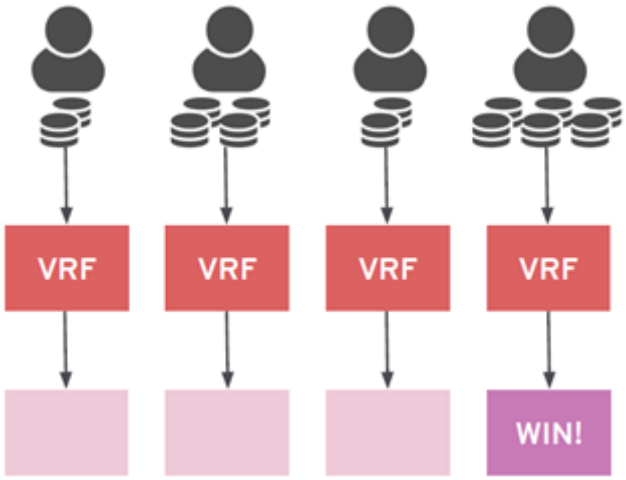
- Indipendente dalla catena pubblica
- Sceglie i propri validatori
- Esegue il proprio algoritmo di consenso Algorand
- Interagisce con la catena principale Algorand per effettuare transazioni con altre co-catene con la stessa facilità e sicurezza
- Gode di tutti i vantaggi della catena principale Algorand

- Su Algorand le trx hanno un costo fisso, indipendentemente dalla trx, perché non ci sono i miner da pagare per il lavoro perché la validazione non consuma nulla.
- L'uso più comune dell'indicizzatore prevede di ottenere blocchi convalidati da un nodo Algorand algod locale, aggiungerli a un database PostgreSQL e servire un'API per rendere disponibile una varietà di query preparate. Alcuni utenti potrebbero voler scrivere direttamente query SQL del database. L'indicizzatore funziona recuperando i blocchi uno alla volta, elaborando i dati del blocco e caricandoli in un database tradizionale. Esiste un livello di astrazione del database per supportare diverse implementazioni del database. Durante il normale funzionamento, il servizio verrà eseguito come demone e richiede sempre l'accesso a un database. Alla fine di luglio 2021, la memorizzazione di tutti i blocchi grezzi in MainNet è di circa 609 GB e il database PostgreSQL di transazioni e conti è di circa 495 GB. Gran parte di quella differenza di dimensioni è l'indicizzatore che ignora i dati della firma crittografica; basandosi su algod per convalidare i blocchi. Trascurando questo, l'indicizzatore può concentrarsi sui dettagli "che cosa è successo" di transazioni e conti.

	Algorand Virtual Machine	Ethereum Virtual Machine
TURING COMPLETENESS	YES	YES
EXECUTION SPEED	~ 4.5 sec regardless dApp complexity	> 20 sec depends on dApp complexity
ENERGY EFFICIENCY	0.000008 [kWh/txn] all final	120 [kWh/txn] not all final
EXECUTION COSTS	~ 0.001 $ (public network) regardless dApp complexity	~ 20 $ (public network) depends on dApp complexity
INTEROPERABILITY	native interoperability ASA, AT, MultiSig, RekeyTo...	user defined (complex) solutions
EFFECTS FINALITY	instant	~ 6 blocks
MATHEMATICAL PRECISION	512 bits	256 bits
PROGRAMMABILITY	TEAL, PyTeal, Reach, ...	Solidity, Viper, Reach, ...

CONSENSO

Algorand utilizza il protocollo di consenso Pure Proof of Stake che prevede una funzione matematica VRF (Verifiable Random Function) eseguita in due fasi; in una prima fase in cui il **nodo vuole proporre il blocco** e in una seconda fase per **la scelta dei validatori** per verificare la validità del blocco

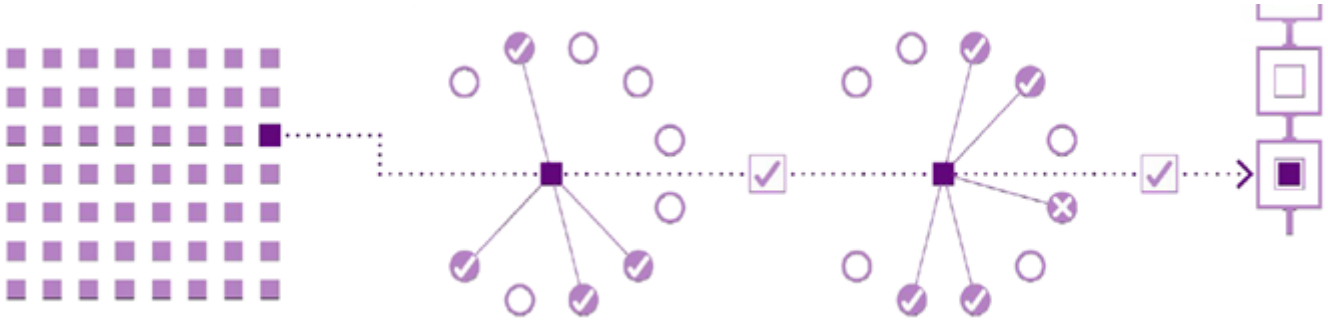


Parametri che dipendono dagli n blocchi precedenti e dallo stato attuale della rete

1 Fase: Nodo -> funzioneVRF(parametri) -> risultato -> Kp(risultato) -> pacchetto dati -> hash(pacchetto)

Se l'hash sta sotto al valore di difficoltà , il nodo calcola il blocco e ci aggiunge il valore hash. Propone il blocco alla rete con l'hash, e tra tutti i nodi che sono riusciti a ricavare l'hash sotto la soglia(difficoltà), la rete sceglie quello che ha l'hash più piccolo.

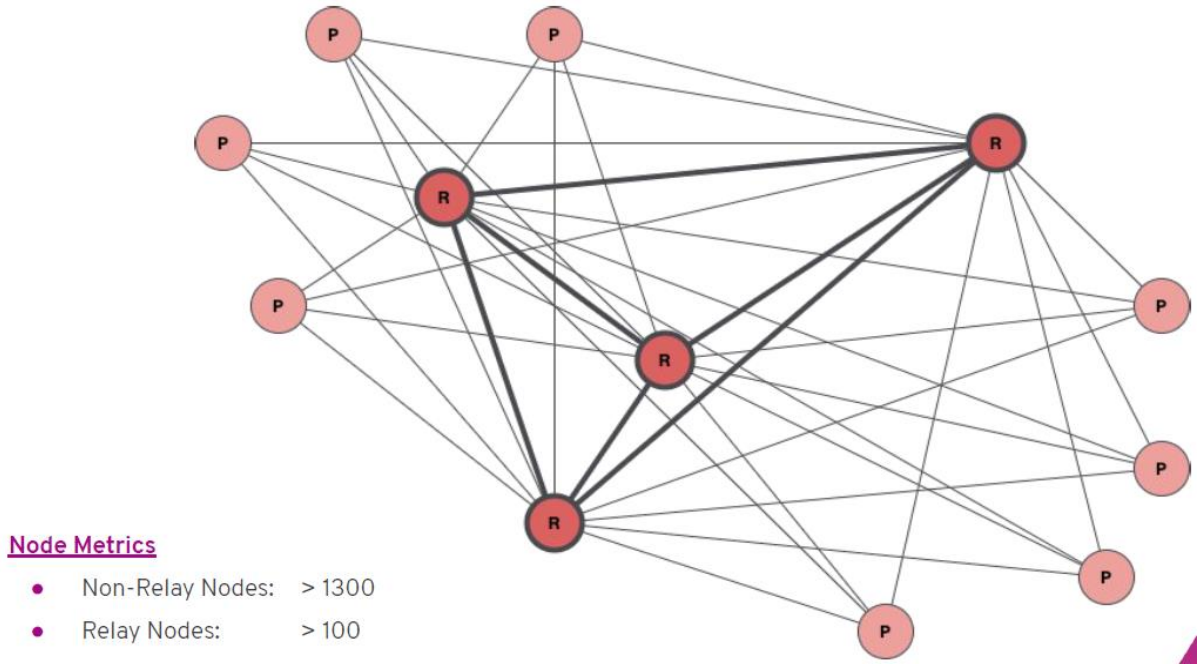
Svantaggio: Un problema può sorgere nel momento in cui per un guasto di rete non vedo il blocco del miner con il numero hash più basso, in questo caso il sistema sceglie un altro miner.



2 Fase: Bisogna selezionare 1000 utenti, che hanno diritto di voto su quel blocco in cui ogni utente decide se il blocco è valido. Come li selezioniamo ? si esegue un'altra funzione VRF, si firma la funzione, guardo il risultato(hash) e se sta sotto un valore dinamico di difficoltà, quel validatore va a far parte di quei mille utenti che possono esprimere il proprio parere su quel blocco. A questo punto, se la maggioranza dei mille utenti scelti, dice che il blocco è valido, il blocco è promosso e può essere aggiunto alla catena.

Dato che l'esecuzione della VRF avviene nel segreto nell'hardware del nodo, e solo dopo mostro il risultato al network, capiamo che un utente malevolo non sa chi corrompere. Nel momento in cui il validatore mostra il risultato, non può essere corrotto perché appunto il validatore si è già esposto alla rete con il risulato.

Svantaggio: In Algorand anche se è chiamata pure proof of stake non c'è uno stake da piazzare e nel caso si bara non c'è uno stake da bloccare.



Node Metrics

- Non-Relay Nodes: > 1300
- Relay Nodes: > 100

L'installazione predefinita imposterà il nodo come nodo non relay.

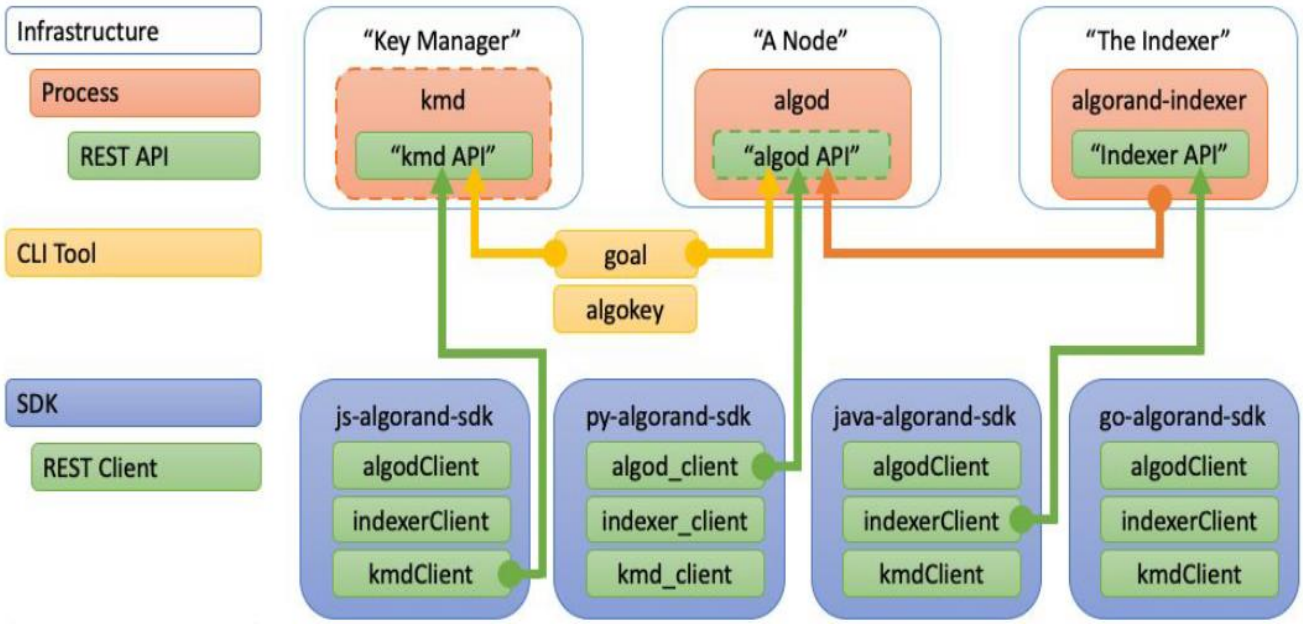
I nodi relè sono sempre impostati in modalità Archiviazione. I nodi non relè hanno la possibilità di essere eseguiti in entrambe le configurazioni.

Un nodo è un nodo di inoltro valido se sono vere due cose:

1. Il nodo è configurato per accettare connessioni in entrata su una porta pubblicamente accessibile (4161 per convenzione).
2. L'indirizzo IP pubblico del nodo (o un nome DNS) e la porta assegnata, sono registrati nei record SRV di Algorand per una rete specifica (MainNet/TestNet).

Pertanto, un nodo di inoltro deve essere in grado di supportare un numero elevato di connessioni e gestire il carico di elaborazione associato a tutti i dati che fluiscono da e verso queste connessioni. Pertanto, i nodi relè richiedono molta più potenza rispetto ai nodi non relè.

ARCHITETTURA



Componenti per interagire con un nodo. L'integrazione con il daemon del protocollo Algorand (algod), il daemon di gestione delle chiavi Algorand (kmd) o il daemon Algorand Indexer (algorand-indexer) viene eseguita utilizzando un set di API REST.

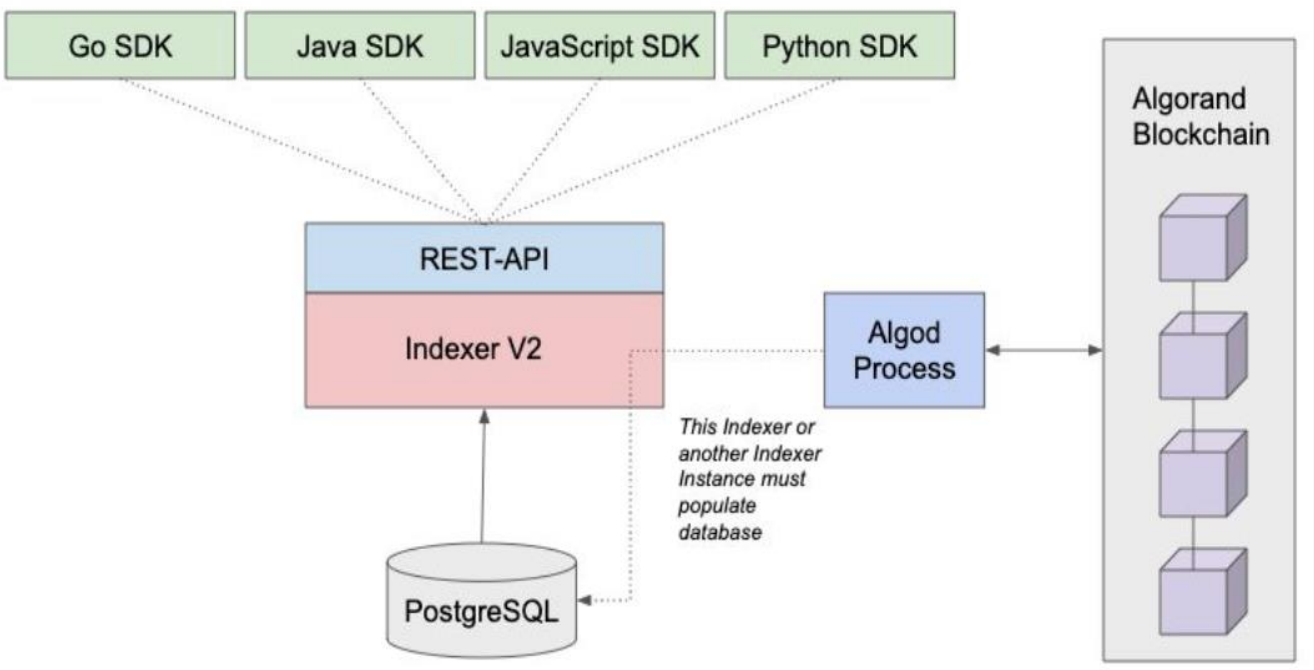
goal è la CLI per interagire e per eseguire operazioni quali: creare un account, elencare registro di chain, esaminare lo stato della rete o creare una transazione, ha anche funzionalità per gestire chiavi, firmare e inviare transazioni, creare asset ed eseguire funzioni disponibili negli SDK.

algod è il principale processo Algorand per la gestione della blockchain; vengono elaborati i messaggi, eseguite alcune fasi del protocollo, blocchi vengono scritti sul disco. . Il processo algod espone anche un API REST che gli sviluppatori possono utilizzare per comunicare con il nodo e la rete. Algod utilizza la directory dei dati per l'archiviazione e le informazioni di configurazione. Un'applicazione si connette alla blockchain di Algorand tramite un client **algod** . Il client algod richiede un **indirizzo IP dell'endpoint REST algod** e **un token algod**

Kmd (key management Deamon) è il demone di gestione delle chiavi. Gestisce interazione con le chiavi private dei clienti, è responsabile della generazione e dell'importazione delle spending key, della firma delle transazioni e dell'interazione con i meccanismi di archiviazione delle chiavi come i portafogli hardware. Questo processo può essere eseguito anche su una macchina separata, isolando le spending key dalla rete. Nella configurazione predefinita, questo sarà nella directory dei dati per algod ma conterrà la propria cartella etichettata kmd-version. Il processo kmd ospita anche un endpoint REST per l'integrazione.

algokey è un'utilità della riga di comando per generare, esportare e importare chiavi. Lo strumento può essere utilizzato anche per firmare transazioni a firma singola e multipla.

INDEXER



L’indexer fornisce un'interfaccia API REST di chiamate API per supportare la ricerca di transazioni, saldi, asset e blocchi. Le API REST dell’indexer recuperano i dati blockchain da un database PostgreSQL. Il database viene popolato utilizzando la stessa istanza dell'indicizzatore o un'istanza separata dell'indicizzatore che deve connettersi al processo algod di un nodo Algorand in esecuzione, per leggere i dati del blocco. Questo nodo deve essere anche un nodo di archiviazione per rendere possibile la ricerca nell'intera blockchain.

L’indexer non fa parte del nodo Algorand e richiede un download binario separato.

- Scalabile — milioni di utenti
- Efficiente — 1000 TPS(transazioni al secondo)
- Velocità di finalizzazione del blocco — < 5s per blocco
- Costo delle transazioni — 0.001 ALGO per transazione (non c’è il lavoro del miner da pagare)
- No soft-fork — probabilita < 10^18 (i 1000 scelgono 1 validatore)
- Transazioni finalizzate — (non attendere le 6 conferme, i 1000 utenti hanno verificato)
- Basse risorse hardware del nodo — (non essendoci il mining ma la VRF semplice funzione)
- Stake senza delega — (non c’è un vero e proprio stake da piazzare)
- No stake minimo per partecipare al consenso
- Carbon negative
- Sicura rispetto ad attacchi DoS

- Resiliente rispetto alla partizione della rete

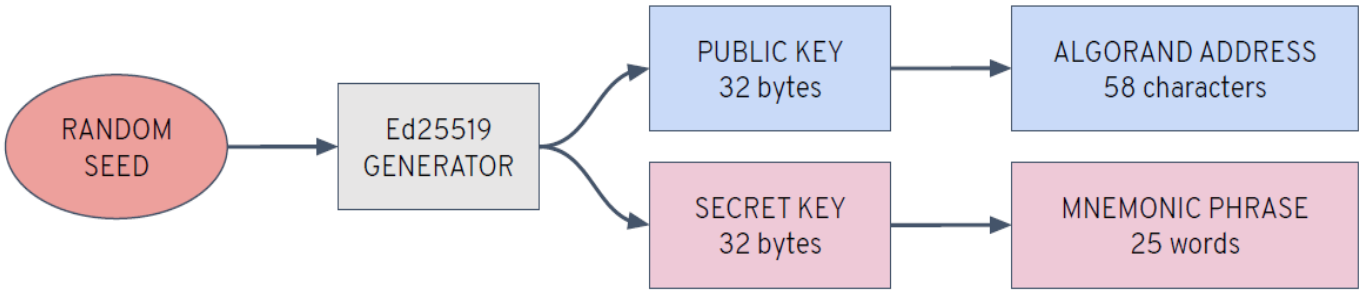
Non si crea una centralizzazione ? chi ha più algo ha più probabilità di parlare ?

Atomic Trasfer: transazioni di gruppo che vengono incollate tra di loro con un hash crittografico. Vengono verificate tutte insieme o nessuna.

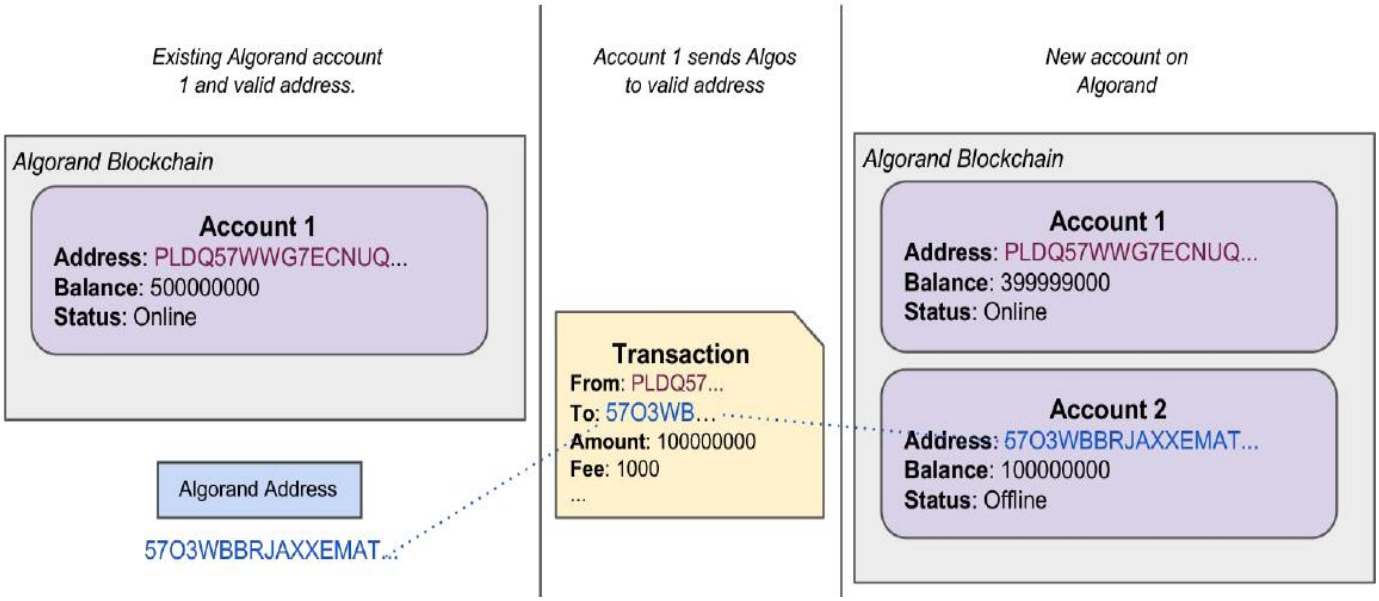
ACCOUNT

Agorand non è un UTXO ma è account base.

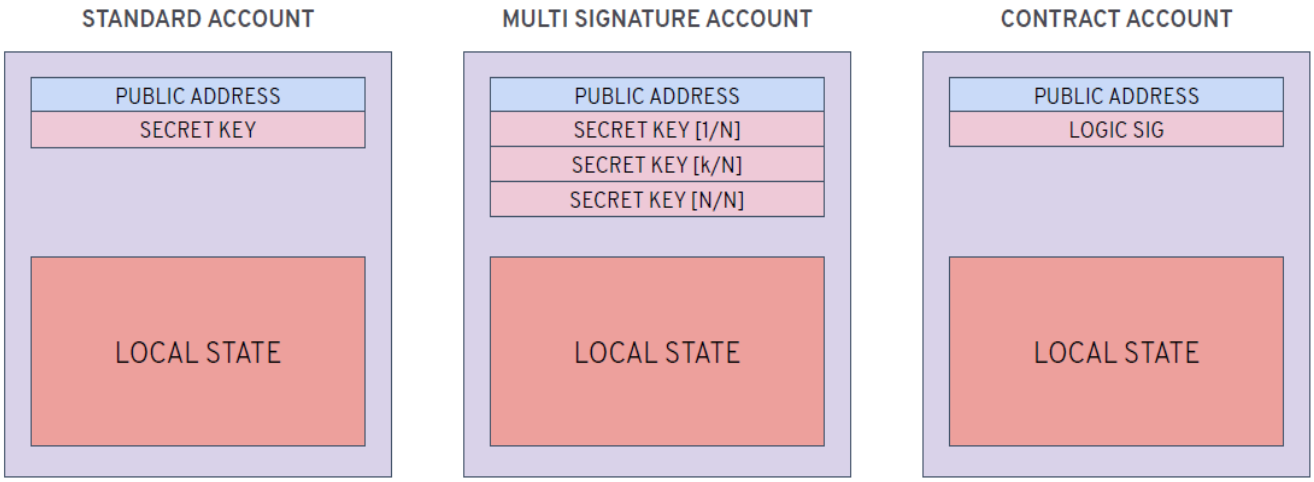
Algorand utilizza le firme a curva ellittica ad alta velocità e sicurezza Ed25519. Le chiavi vengono prodotte tramite librerie crittografiche open source standard fornite con ciascuno degli SDK. L'algoritmo di generazione della chiave prende un valore casuale come input ed emette due array da 32 byte, che rappresentano una chiave pubblica e la chiave privata associata. Questi sono anche indicati come coppia di chiavi pubblica/privata. Queste chiavi svolgono importanti funzioni crittografiche come la firma dei dati e la verifica delle firme.



La generazione delle chiavi può avvenire anche off-line. Le chiavi generate entrano a far parte di Algorand, inviando un prima quantità di Algo (minimo 0,1 Algo) sulla quella chiave pubblica, essa sarà associata ad Algorand. Ora la chiave pubblica, generata precedentemente off-line, viene associata ad un account.



Stato locale, in cui sono salvate tutte le info che sono associate a quella chiave pubblica(balance token, balance algo, variabili smart-contract tutto quello che ha che fare con il nostro account specifico). Lo stato locale appartiene univocamente a quella chiave pubblica e le uniche modifiche possono essere usando la chiave segreta.



Multisignature Account: un account non è necessariamente controllato da un sola persona con la sua chiave segreta associata, ma può essere anche controllato da più persone quindi più chiave segrete. Può essere impostato un potere di firma ad esempio 2 su 4, cioè se almeno ho 2 firme su quella trx, la trx viene approvata.
Contract Account o smart contract stateless: ha un chiave pubblica. Tutte le azioni che vengono fatte, non sono firmate da una chiave privata ma sono firmate da una logica/programma.
Primitiva **Rekeying Algorand**: mantenendo la stessa chiave pubblica possiamo cambiare le chiavi private.

SMART CONTRACT

LAYER-1

Smart Contract eseguibile su layer-1 è trattato come una transazione, con le stesse caratterestiche:

- Stessa velocità, scalabilità e sicurezza
- Stessi costi
- Transazione finalizzata
- Cosuma poco
- Esecuzione e validazione sm avviene durante la formazione del blocco.
- Le smart-contract call, come le trx call, finiscono nel blocco generato ogni 4,4. Ciò significa che un esecuzione dello sm non rallenta tutta la catena come in altre BC.
- Gli sm su layer-2, rispetto a quelli su layer-1, risolvono il problema di una necessità computazionale o di grandezza di storage più espansiva rispetto agli sm di oggi.
- Sm, essendo su layer 1, preservano un alto livello di concorrenzialità in esecuzione, quindi senza bloccare l'intero ecosistema
- I contratti on-chain sono molto leggeri da eseguire perché sono allo stesso livello della stessa moneta ALGO, quindi i nodi possono eseguirli quasi in tempo reale.
- Smart contract (applicazioni decentralizzati) hanno accesso sia alle info stateless sia stateful, leggono e scrivono lo stato della chain.
- L'unica cosa che la logica dei nostri contratti intelligenti può fare è decidere se le transazioni che coinvolgono il contratto devono fallire o avere successo. Un contratto non può inviare transazioni da solo.

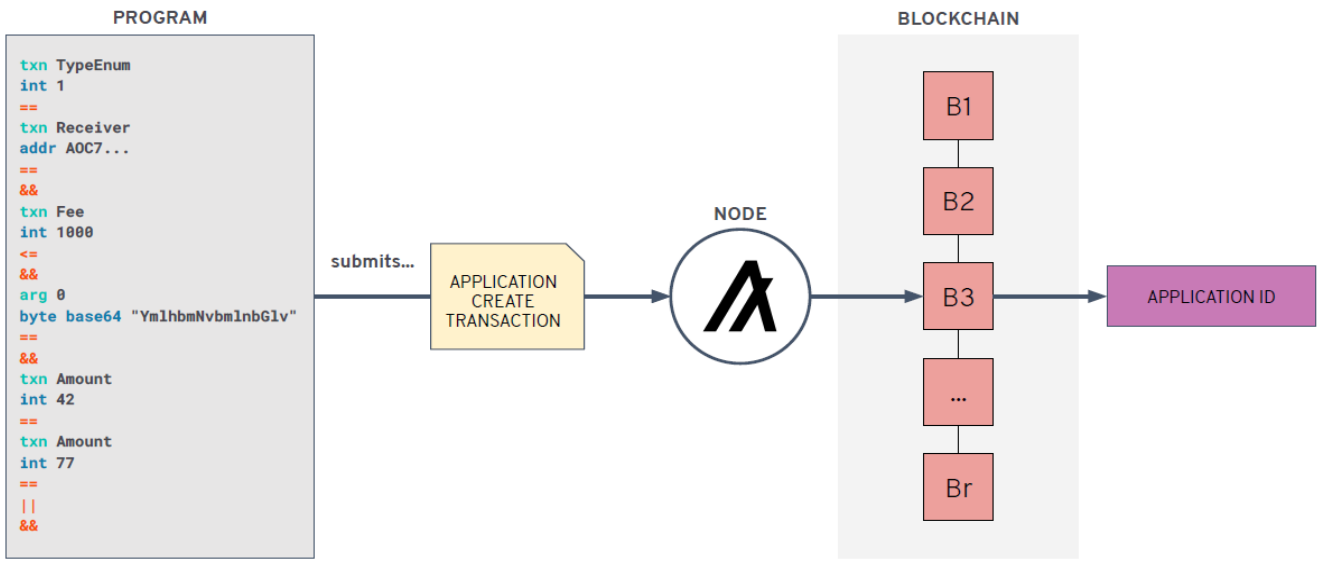
Gli smart contract sono spesso indicati come applicazioni su Algorand. Una volta scritto, un contratto viene distribuito utilizzando una transazione **create application**, se quest'ultima va a buon fine ho un application ID. Nell'application ID è memorizzata la logica del programma Teal (sm). Gli smart-contract sono richiamabili da remoto tramite le **application call,** che se fatte al contratto valideranno le trx e interazioni in base alla logica Teal.

Gli smart contract sono richiamabili da remoto tramite un tipo di transazione speciale chiamato **application transaction**.

Il codice di una dApps può separare il codice del contratto intelligente e le operazioni SDK in due file separati: **contracts.py** contiene tutta la logica per lo smart contract e **operations.py** contiene il codice SDK per distribuire lo smart contract e comunicare con esso una volta distribuito.

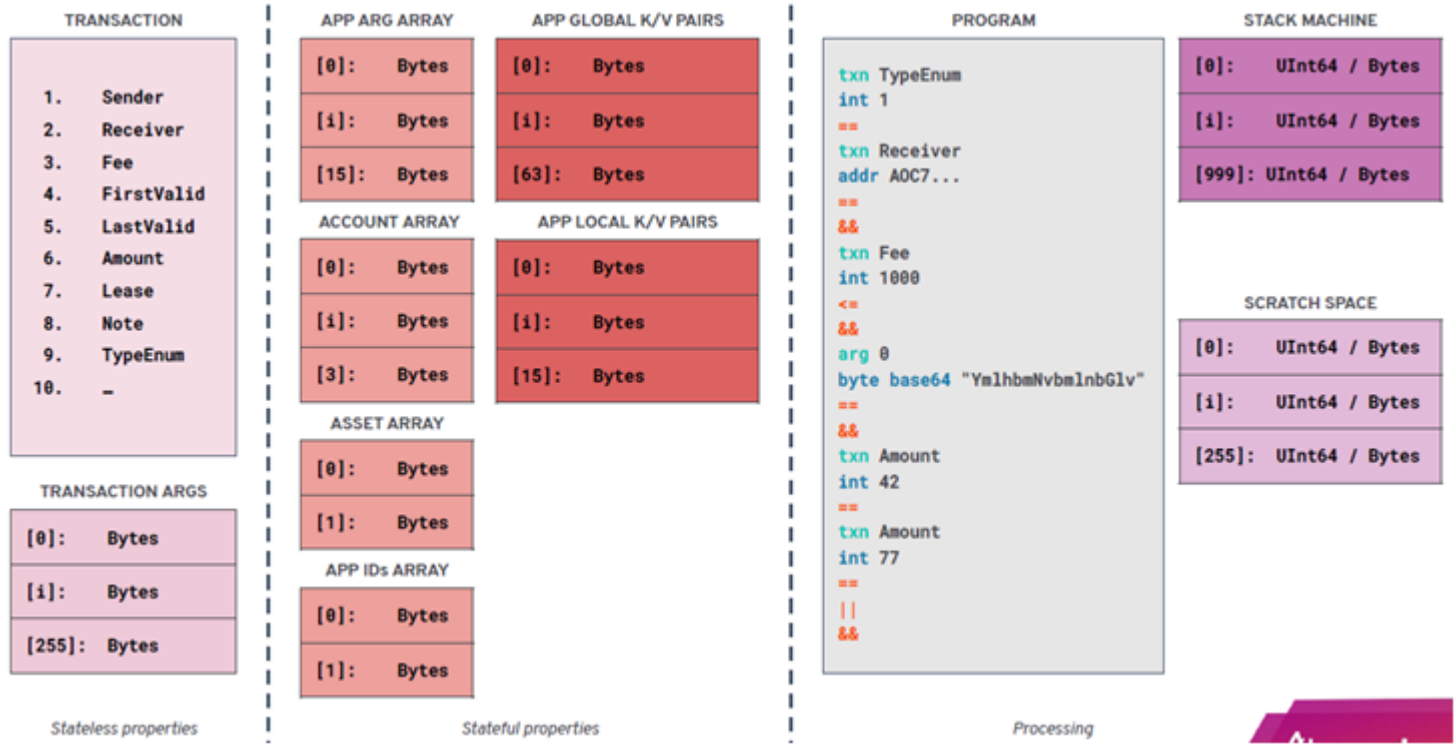
Panoramica PyTeal
Scriviamo lo smart-contract utilizzando la libreria Pyteal e python, poi il compilatore python compila il codice “scritto in pyteal”, ricavando automaticamente il contratto scritto in Teal. Il Teal sarà inviato alla rete.

Gli smart contract sono in realtà composti da due "programmi" separati.
Approval program: contiene la maggior parte del codice e avrà esito positivo solo se nello stack viene lasciato un valore diverso da zero al completamento del programma o se il return del codice operativo viene chiamato con un valore positivo in cima allo stack.
Clear Program: Viene utilizzato per gestire gli account utilizzando la chiamata clear per rimuovere lo smart contract dal record del saldo. Questo programma passerà o fallirà allo stesso modo ApprovalProgram. In PyTeal approval program e clear program sono generalmente creati nello stesso file Python.



Gli smart contract possono anche memorizzare valori sulla blockchain. Questa memoria può essere globale o locale.

L'archiviazione locale si riferisce all'archiviazione di valori in un record di saldo contabile. Lo storage globale è un dato che viene specificamente archiviato sulla blockchain
??????????????????????????????



Smart contract hanno accesso sia alle info stateless sia stateful, leggono e scrivono lo stato della chain.
Quindi le informazioni nel file TEAL(smart contract) sono prese da due spazi differenti:
Stateless: sono quei oggetti/variabili presentati dalle nuove transazioni, ad esempio le transazioni che si devono approvare prima che finiscono nel ledger.
Stateful: sono info di stato cioè che sono già state scritte nel ledger. Come il balance di un account, variabili di uno smart-contract

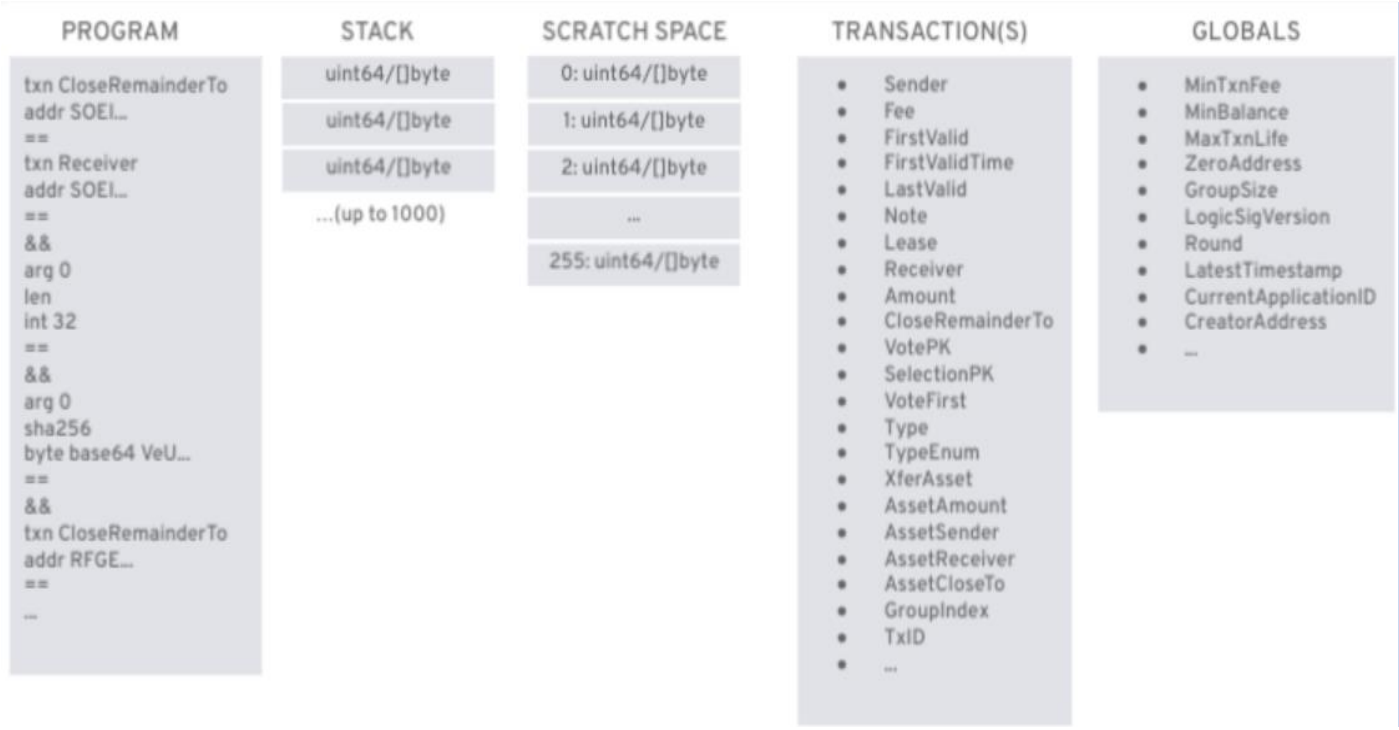
Questi contratti hanno una serie di funzioni (codici operativi della libreria pyteal) che possono essere richiamate. Utilizzano dati on-chain come saldi, argomenti aggiuntivi, transazioni aggiuntive e valori memorizzati per essere valutati come veri o falsi. Inoltre, come parte della logica, **le variabili on-chain possono essere memorizzate per contratto o per account.**

Panoramica PyTeal
Scriviamo lo smart-contract utilizzando la libreria Pyteal e python, poi il compilatore python compila il codice “scritto in pyteal”, ricavando automaticamente il contratto scritto in Teal. Il Teal sarà inviato sulla rete.
In PyTeal approval program e clear program sono generalmente creati nello stesso file Python.

OnComplete

Le chiamate agli smart contract vengono implementate tramite le trx ApplicationCall. Questi tipi di transazione sono i seguenti:

- NoOp - Chiamate generiche dell'applicazione per eseguire l’ApprovalProgram
- OptIn: gli account utilizzano questa transazione per attivare lo smart contract per partecipare (utilizzo dello spazio di archiviazione locale).
- DeleteApplication - Transazione per eliminare l'applicazione.
- UpdateApplication - Transazione per aggiornare i programmi TEAL per un contratto.
- CloseOut - Gli account utilizzano questa transazione per chiudere la loro partecipazione al contratto. Questa chiamata può fallire in base alla logica TEAL, impedendo all'account di rimuovere il contratto dal suo record di saldo.
- ClearState - Simile a CloseOut, ma la transazione cancellerà sempre un contratto dal record del saldo del conto indipendentemente dal fatto che il programma abbia esito positivo o negativo.



I programmi TEAL vengono elaborati una riga alla volta, inserendo e rimuovendo i valori dallo stack

TEAL consente/detiene:

- di passare argomenti al programma da una transazione,
- ha uno spazio vuoto per memorizzare temporaneamente valori da utilizzare successivamente nel programma,
- accedere a proprietà di transazione singole o raggruppate,
- accedere a valori globali,
- ha un paio di pseudo operatori, costanti e funzioni di controllo del flusso.

Gli smart contract possono leggere e scrivere l'archiviazione globale per il contratto e l'archiviazione locale per gli account che aderiscono al contratto

Alcuni degli opcode in TEAL sono validi solo per un tipo di contratto specifico, indicando il tipo nell'attributo **Mode**. Questo attributo sarà impostato **Signature** per smart signature e **Application** per smart-contract. Ad esempio, la lettura degli asset del conto o dei saldi di Algo è disponibile solo negli smart contract.

Lo scopo principale di un programma TEAL è restituire true o false. Al termine del programma, se nello stack è presente un valore diverso da zero, restituisce true. Se è presente un valore zero o lo stack è vuoto, restituirà false. Se lo stack ha più di un valore, anche il programma restituisce false a meno che non venga utilizzato l'opcode return.

Il programma utilizza il codice operativo pyteal Txn per fare riferimento all'elenco delle proprietà della transazione corrente. Le proprietà delle transazioni raggruppate sono referenziate utilizzando Gtxn e Gtxns. Il numero di transazioni in una transazione raggruppata è disponibile nella variabile globale GroupSize. Per ottenere il destinatario della prima transazione utilizzare Gtxn o Receiver.

Lo addr pseudo codice operativo converte gli indirizzi Algorand in una costante di byte e inserisce il risultato nello stack.

Ad esempio, l' == operatore valuta se gli ultimi due valori nello stack sono uguali e inserisce 1 o 0 a seconda del risultato. Il numero di valori utilizzati da un operatore dipenderà dall'operatore

TRANSAZIONI

There are six transaction types in the Algorand Protocol:

- Payment
- Key Registration
- Asset Configuration
- Asset Freeze
- Asset Transfer
- Application Call

Application Call Transaction

An Application Call Transaction is submitted to the network with an **AppId** and an **OnComplete** method. AppId specifica quale app chiamare e il metodo OnComplete viene utilizzato nel contratto per determinare quale ramo della logica eseguire.

Le Application Call possono includere altri campi necessari alla logica come:

- ApplicationArgs - Per passare argomenti arbitrari a un'applicazione (o in futuro per chiamare un metodo ABI)
- Accounts: per trasferire account che potrebbero richiedere il controllo del saldo o lo stato di attivazione
- ForeignApps: per passare le app e permettere l'accesso allo stato a un'applicazione esterna (o in futuro per chiamare un metodo ABI)
- ForeignAssets - Per passare gli ASA per il controllo dei parametri.
-

Application Create Transaction

Quando deve essere creata un'applicazione, il metodo **OnComplete** è impostato su **NoOp**, nessun AppId è impostata e vengono passati i programmi approval/clear e lo Schema delle variabili locali e globali. Il programma approval può eseguire ulteriori controlli durante l'installazione verificando che AppId == 0.

Supponendo che tutti i controlli del saldo e della firma vengano superati, verrà creata un'applicazione con un nuovo AppId.

Application Update Transaction.

Una Application Update Transaction può essere inviata e approvata presupponendo che la logica dell'approval program lo consenta. Ciò avviene specificando l'AppId da aggiornare e passando la nuova logica per i Approval and Clear programs.

Application Delete Transaction

Una domanda può essere cancellata fintanto che la logica nel Programma di approvazione lo consente.

Application Opt-In Transaction

Una transazione di partecipazione(Opt-In) all'applicazione deve essere inviata da un account affinché lo stato locale possa essere utilizzato per quell'account. Se non è richiesto alcuno stato locale, questa transazione non è necessaria per un dato account.

Application Close Out Transaction

Una transazione di chiusura dell'applicazione viene utilizzata quando un account vuole recedere da un contratto e rimuovere il suo stato locale dal suo balance record. Questa transazione potrebbe non riuscire secondo la logica dell'approval program.

Application Clear State Transaction

Una transazione Application Clear State viene utilizzata per forzare la rimozione dello stato locale dal record di saldo del mittente. Data una transazione ben formata, questo metodo avrà sempre successo. Il programma Clear viene utilizzato dall'applicazione per eseguire qualsiasi tenuta della contabilità necessaria per rimuovere l'account dai suoi archivi.

Application NoOp Transaction

Le transazioni Application NoOp costituiscono in pratica la maggior parte dei metodi di Application Call. La logica in uno smart contract spesso si ramifica alla logica appropriata dato il contenuto dell'array ApplicationArgs passato.

ASA

Il protocollo Algorand supporta la creazione di risorse on-chain che beneficiano della stessa sicurezza, compatibilità, velocità e facilità d'uso di Algo. Con Algorand Standard Assets si può rappresentare stablecoin, punti fedeltà, crediti di sistema e punti in-game, solo per citare alcuni esempi. Puoi anche rappresentare singole risorse uniche come un atto per una casa, oggetti da collezione, parti uniche di una catena di approvvigionamento, ecc. C'è anche una funzionalità opzionale per porre restrizioni di trasferimento su una risorsa che aiutano a supportare i casi d'uso di titoli, conformità e certificazione.

Ecco alcune cose da tenere presenti prima di iniziare con gli assets:

- Un singolo account Algorand può creare fino a 1000 asset.
- Per ogni asset creato o posseduto da un account, il suo saldo minimo viene aumentato di 0,1 Algo (100000 microAlgo).
- Prima che un nuovo assets possa essere trasferito su un conto specifico, il destinatario deve acconsentire a ricevere l'asset.
- Se viene emessa una transazione che violerebbe il numero massimo di attività per un account o non soddisferebbe i requisiti di saldo minimo, la transazione avrà esito negativo.

Asset parameters

Il tipo di asset creato dipenderà dai parametri che vengono passati durante la creazione dell asset e talvolta durante la riconfigurazione dell'asset.

Parametri delle risorse immutabili. Questi otto parametri possono essere specificati solo quando viene creata una risorsa:

- Creator (required)
- AssetName (optional, but recommended)
- UnitName (optional, but recommended)
- Total (required)
- Decimals (required)
- DefaultFrozen (required)

- URL (optional)
- MetaDataHash (optional)

Parametri delle risorse mutabili. Esistono quattro parametri che corrispondono a indirizzi che possono autorizzare funzionalità specifiche per un asset. Questi indirizzi devono essere specificati al momento della creazione ma possono anche essere modificati dopo la creazione. In alternativa, questi indirizzi possono essere impostati come stringhe vuote, che bloccheranno irrevocabilmente la funzione su cui avrebbero avuto autorità. Ecco i quattro tipi di indirizzo:

Manager Address: è l'unico account che può autorizzare le transazioni a riconfigurare o distruggere un asset.

Reserve Address: Specificare un account di riserva significa che gli assets non coniati risiederanno in tale account anziché nell'account creatore predefinito. Gli asset trasferiti da questo conto sono unità "coniate" dall'asset.

Freeze Address: L'account di blocco può bloccare o sbloccare le disponibilità di asset per un account specifico. Quando un account è bloccato, non può inviare o ricevere l'asset bloccato.

Clawback Address: L'indirizzo di clawback rappresenta un conto autorizzato a trasferire asset da e verso qualsiasi detentore di asset (supponendo che abbiano aderito). Usalo se hai bisogno dell'opzione per revocare asset da un account (come se violano determinati obblighi contrattuali legati alla detenzione del bene).

FRONTEND

CICLO DI VITA APPLICAZIONE

Il ciclo di vita di un app; dalla creazione iniziale, all'utilizzo, alla modifica e infine all'eliminazione. L'applicazione memorizza il numero di volte in cui è stata chiamata all'interno del suo ***stato globale*** e memorizza anche il numero di volte in cui ciascun account utente chiama l'applicazione all'interno del proprio ***stato locale***. A metà del ciclo di vita, l'applicazione viene aggiornata per aggiungere un'ulteriore coppia chiave:valore *all'archiviazione locale* dell'utente per l'archiviazione del timestamp della chiamata.

State storage

Definiamo lo schema globale e locale. Questi valori sono immutabili una volta che l'applicazione è creata, quindi loro specificano il numero massimo richiesto all'inizio dell applicazione e per i futuri aggiornamenti.

Create:

The creator will deploy the application using the `create app` method.

Set the on_complete parameter to NoOp: `on_complete =`

`transaction.OnComplete.NoOpOC.real`

Opt-in:

The user must opt-in to use the application.

Construct the transaction with defined values: `txn = transaction.ApplicationOptInTxn(sender, params, index)`

Call (NoOp):

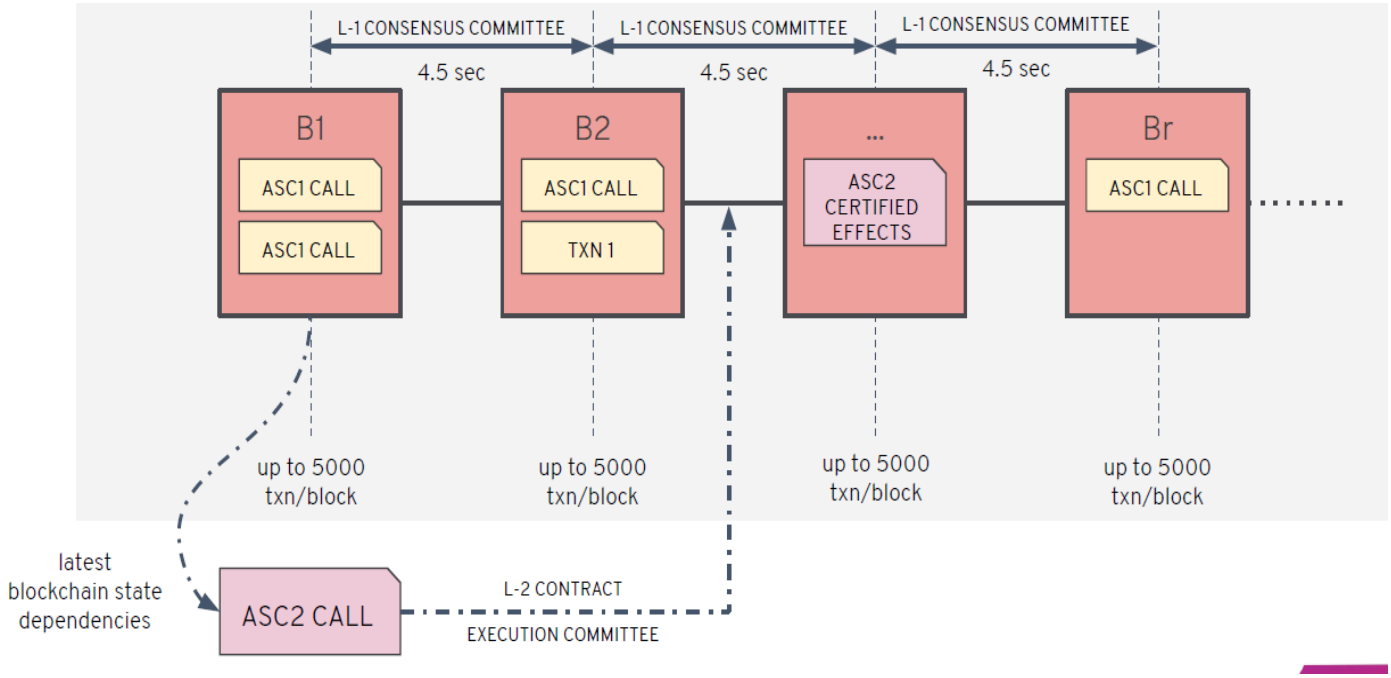
The user may now call the application: `txn = transaction.ApplicationNoOpTxn(sender, params, index)`

Update:

SMART-CONTRACT

SMART-CONTRACT LAYER-2

Se in futuro ci sarà parecchia complessità computazionale si prende un ramo diverso dal diretto consenso e si fa: 1 una fotografia alla chain 2 mi sposto in una computazione in parallelo in cui c'è un comitato del layer-2 (cioè ci sono dei validatori che certifica che l'esecuzione ha dato lo stesso risultato agli altri partecipanti). 3 quando si trova un accordo che l'esecuzione è valida, allora gli effetti della computazione vengano registrati sul layer-1. Ciò significa che se ero partito con alcune dipendenze dello stato della chain su L1, dopo aver eseguito la computazione, se i risultati non sono in disaccordo con lo stato di partenza della chain, allora gli effetti sono certificati in L1, altrimenti no. L'output cioè i risultati dell'esecuzione dello sm, vengono inviati come trx standard allo L1.



Se gli input cambiano tra il momento in cui la transazione è stata originariamente trasmessa e il momento in cui gli output vengono reinviati alla L1, il nodo di convalida annulla la transazione o invia nuovamente la transazione con i nuovi input. In questo modo, i blocchi possono avvenire in modo asincrono e possono essere elaborati più contratti "pesanti" contemporaneamente.
Al contrario, i contratti on-chain sono molto leggeri da eseguire perché sono allo stesso livello della stessa moneta ALGO, quindi i nodi possono eseguirli quasi in tempo reale.

Layer-2 differenze con Ethereum

Scambio tra Alice e Bob. Alice vuole essere sicura che se trasferisce le monete, ottiene i gettoni e Bob vuole rassicurazioni simili. Questo tipo di transazione, in cui i trasferimenti avvengono entrambi o non avvengono entrambi, è chiamato atomic swap. In Ethereum richiede un contratto di blocco temporale con hash in cui qualsiasi errore di programmazione può essere un problema. Al contrario, gli smart contract Algorand Layer-1 forniscono una soluzione semplice e sicura agli scambi atomici, mettendo a disposizione le Atomic Trasfer.

Nei contratti off-chain di Algorand tot. utenti che devono usare lo stesso smart-contract non devono aspettare che il precedente utente finisce ma possono usarlo contemporaneamente.

Gli smart contract di Ethereum forniscono supporto integrato per la propria valuta Ether, ma i clienti che desiderano creare il proprio token simile a una valuta non sono agevolati. Sebbene gli standard e le convenzioni si siano evoluti per i token definiti da Ethereum, scrivere tale codice può ancora essere rischioso e ci sono stati attacchi riusciti ai token definiti dall'utente in Ethereum.

L'architettura del contratto intelligente di Algorand, al contrario, fornisce il supporto integrato per Algorand Standard Assets definiti dall'utente, allo stesso livello della valuta Algo nativa di Algorand . La blockchain di Algorand fornisce una protezione integrata contro la creazione o l'eliminazione involontaria di token.

Ora supponiamo di avere tot. utenti che usano lo stesso smart contract, ognuno deve aspettare che l'utente precedente finisce le operazioni. Ogni esecuzione di smart contract di Ethereum blocca l'avanzamento della blockchain nel suo insieme, limitando la scalabilità per la creazione di un nuovo blocco. Nei contratti off-chain di Algorand tot. utenti che devono usare lo stesso smart contract non devono aspettare che il precedente utente finisce ma possono usarlo contemporaneamente.
I contratti off-chain possono leggere i saldi dei conti e altre informazioni sulla catena principale e possono emettere transazioni, come quelle per i pagamenti, che modificano lo stato della blockchain. A differenza dei contratti convenzionali in stile Ethereum, queste "transazioni di effetti" non vengono eseguite direttamente. Invece, gli effetti della chiamata sono convalidati da un quorum di validatori del comitato di esecuzione del contratto. Le transazioni della chiamata sono impacchettate in un batch di transazioni "tutto o niente" di livello 1, per avere insieme successo o meno.

SMART SIGNATURE

Le smart signatures contengono la logica utilizzata per firmare le transazioni. La logica degli smart signatures viene inviata con una transazione. Anche se la logica nello smart signatures sia archiviata nella catena come parte della risoluzione della transazione, la logica non è richiamabile da remoto. Qualsiasi trx, che si basa sullo stesso smart signatures, reinvierà la logica.

Quando la logica viene inviata a un nodo, l'AVM valuta la logica, dove fallisce o riesce. Se la logica di una firma intelligente non riesce, la transazione associata non verrà eseguita.

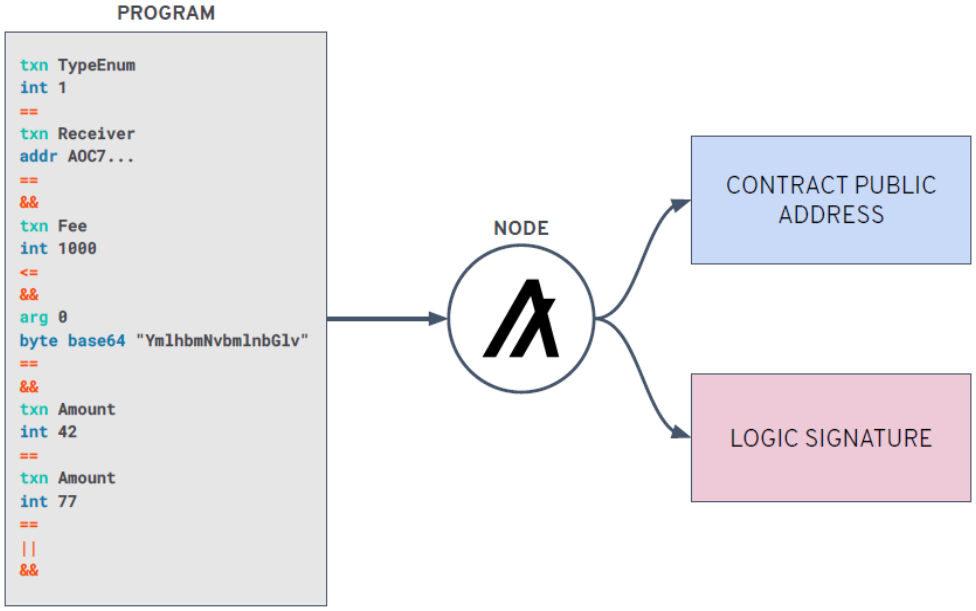
Quando le smart signatures compilate producono un account Algorand che funziona in modo simile a qualsiasi altro account sulla blockchain. Questi account possono contenere algoritmi o asset. Questi fondi possono lasciare l'account solo se si verifica una transazione dall'account che esegue correttamente la logica all'interno dello smart signatures. E' simile nella funzionalità a uno smart contract escrow, ma la logica deve essere inviata per ogni transazione dall'account.

Una volta inviata, una transazione firmata con una smart signatures viene valutata da un nodo Algorand utilizzando la macchina virtuale Algorand. Questi contratti hanno accesso solo ad alcune variabili globali, ad alcuni spazi temporanei e alle proprietà delle transazioni con cui vengono inviati.

Le smart signatures possono anche essere utilizzate per delegare una parte dell'autorità a un altro account. In questo caso, un accountA delega l'accountB. L'accountA può firmare la smart signatures che può quindi essere utilizzata dall'accountB in un secondo momento per firmare una transazione per mezzo dell'accountA.

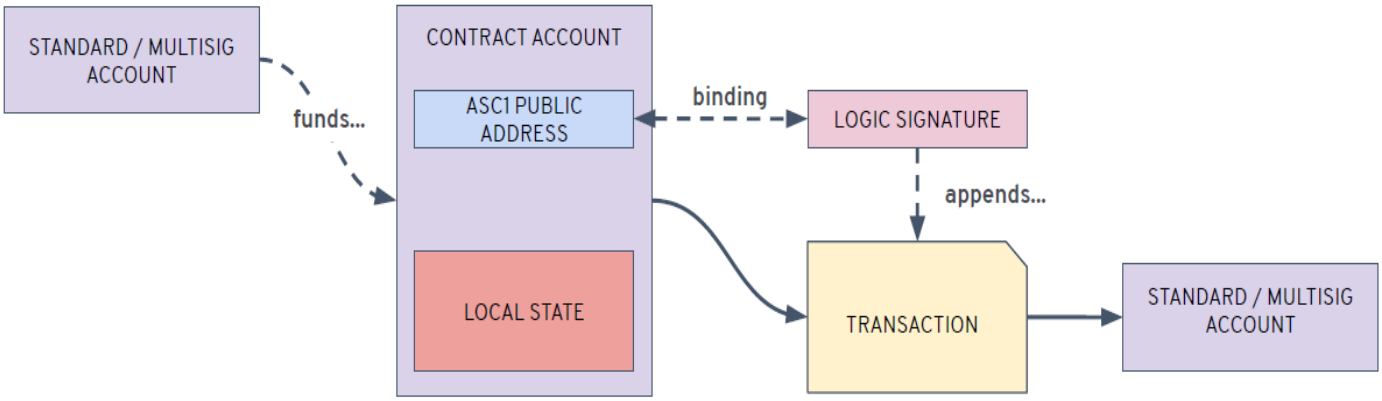
Smart signature hanno accesso solo alle info stateless, ne leggono ne scrivono ma sono firme logiche, che approvano o rifiutano delle transazioni in base a delle condizioni logiche al suo interno. Quindi sono programmi la cui logica governa l'autorizzazione delle transazioni.

Supponiamo che abbiamo un programma Teal e che lo vogliamo affidare ad uno smart signature, cioè un indirizzo che approva o rigetta transazioni solo in base alla logica nel programma Teal. Come si fa ?

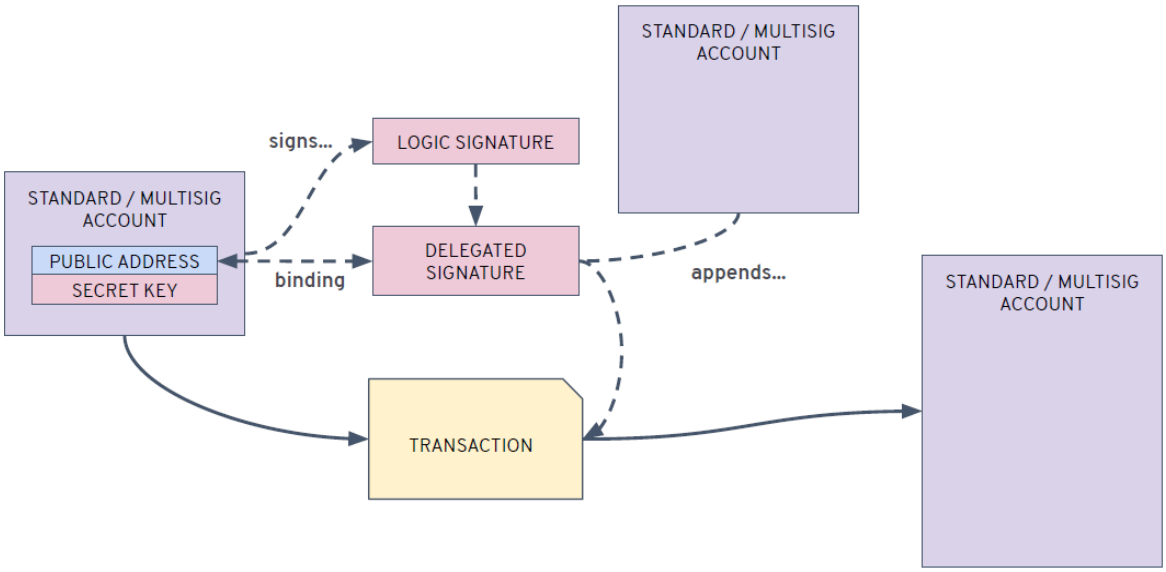


Il nodo compila il Teal e crea due oggetti legati tra loro: l'indirizzo pubblico e lo smart signature, entrambi associati al contratto Teal. L'indirizzo pubblico è l'hash del codice sorgente del programma Teal (quindi non posso modificare nessun istruzione del contratto senza che si modifichi l'hash chiave pubblica). Quindi la chiave pubblica ha un legame univoco con la logica del Teal e la chiave pubblica è associata una firma logica.

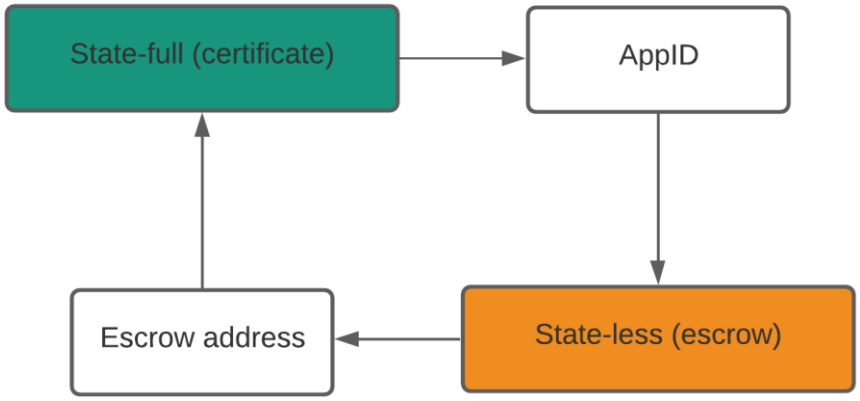
Ho un contratto che è in binding 1 a 1 con la logica (ss) del programma e la chiave pubblica del contratto. Se dico al contratto di darmi un token, la trx appena creata sarà firmata dal contratto con la key pubblica, verrà firmata dallo ss se e solo se vengono rispettate le condizioni dello ss. Quindi è un approvazione basata sulla logica.



Delegated Signature
Posso dire ad una ss di non utilizzare la key pubblica, ma scrivo un smart contratto firmato con la mia chiave privata ottenendo una delegated signature cioè una firma che in binding con la key pubblica. Serve a delegare parte delle operazioni che avvengono dal mio account ad uno smart contract.



CASI D'USO



Il nostro dapp sarà composto da due contratti, un contratto stateless e uno stateful. Su Algorand solo i contratti stateless possono detenere fondi. Se vogliamo archiviare temporaneamente le offerte durante l'asta, dovremo accoppiare uno smart contract stateless al nostro contratto stateful come escrow(deposito a garanzia). Per accoppiare i due dovremo archiviare l'app id del nostro contratto stateful nell'escrow. E aggiungi l'indirizzo escrow nel contratto con stato.