SAPIENZA
UNIVERSITÀ DI ROMA

Studio e realizzazione di smart contract su piattaforma Algorand

Facoltà di Ingegneria dell'Informazione e Statistica

Corso di Laurea in Informatica

Candidato

Marco Raffaele
Matricola 1799912

Relatore

Prof. Claudio Di Ciccio

Anno Accademico 2020/2021

Studio e realizzazione di smart contract su piattaforma Algorand
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: raffaele.1799912@studenti.uniroma1.it

Indice

Capitolo 1

Introduzione

I sistemi basati sulla finanza centralizzata (CeFi) attualmente diffusi, come ad esempio le banche, sono composte da sistemi separati. Ciò significa che ogni singola entità che ne fa parte ha il completo controllo sulla registrazione, archiviazione e gestione dei propri dati. Queste entità inoltre possono scegliere e gestire in maniera totalmente indipendente quali server utilizzare, di quale tipo, e i relativi protocolli di sicurezza. Da quanto appena detto è evidente che questi sistemi risultino essere separati tra loro, il che porta ovviamente ad un'alta difficoltà di comunicazione e di integrazione. Pertanto, il trasferimento di valori e risorse tra questi diversi sistemi centralizzati risulta essere altamente inefficiente sia in termini di costi sia in termini di tempistiche.
Nel campo dell'economia la tecnologia che ha avuto la maggior risonanza negli ultimi decenni è sicuramente la blockchain, ovvero, una tecnologia a registro distribuito (Distribuited Ledger Technology, DLT). Una DLT è una struttura dati che consente di mantenere informazioni in modo completamente trasparente, sicuro, decentralizzato, e soprattutto in maniera immutabile. Questo significa che non è possibile manipolare in alcun modo i dati presenti all'interno blockchain. La trasparenza dei dati viene garantita dal fatto che le informazioni registrate nel ledger (il registro) sono tutte completamente accessibili online. Infine l'immutabilità dei dati memorizzati viene garantita mediante l'uso di diverse tecniche di hashing e della crittografia, il che rende la manomissione di tali dati impossibile. Ogni blocco mantiene diverse informazioni espresse sotto forma di record transazionali e possiede un valore di hash corrispondente al blocco che lo precede all'interno della catena. Questo valore consente di identificare un blocco all'interno della blockchain in maniera univoca. Eventuali modifiche alle informazioni contenute all'interno dei blocchi portano ad una conseguente modifica del valore di hash lungo tutta la catena. Di conseguenza, questo tipo di connessioni, unitamente all'uso di chiavi crittografate, è ciò che rende la blockchain assolutamente sicura.

È proprio sulla tecnologia blockchain che si sta sempre di più diffondendo la così detta finanza decentralizzata, o DeFi, ovvero una nuova concezione di finanza che pone come suo obiettivo primario la riduzione o la totale eliminazione di intermediari nelle operazioni finanziarie tramite l'uso di reti decentralizzate. La DeFi mira a creare un sistema finanziario aperto e che non richieda fiducia agli utenti che ne prendono parte, poiché tale fiducia viene garantita attraverso codice informatico [1]. La finanza decentralizzata consente agli utenti di avere il pieno e diretto controllo dei propri investimenti e delle proprie risorse finanziarie.
Alla base di questo nuovo modello finanziario ci sono quelli che vengono chiamati smart contract, ossia una traduzione di clausole contrattuali, che coinvolgono più parti interessate, codificate in linguaggio informatico, con l'obiettivo di garantire il mantenimento di tali clausole in maniera totalmente autonoma e sicura e al verificarsi di determinate condizioni. Gli smart contract saranno il tema principale di questo elaborato e verranno ampiamente descritti nei prossimi capitoli.

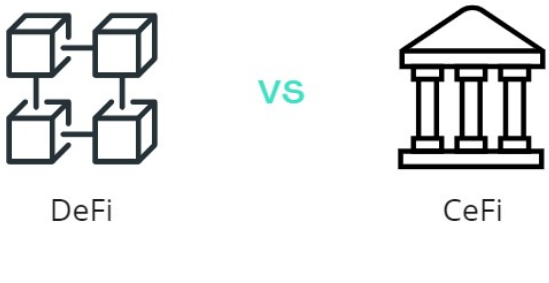

Figura 1.1. DeFi vs CeFi

1.1 Tecnologie DeFi

Tra le tecnologie più diffuse che fanno del loro punto di forza la DeFi troviamo Ethereum [10]. Ethereum nasce nel 2015 ed è una piattaforma open-source per la creazione di applicazioni decentralizzate, anche dette DApp. Ethereum è anche la blockchain più diffusa dal punto di vista dello sviluppo di smart contract i quali vengono generati attraverso un linguaggio di programmazione Turing-completo, chiamato Solidity.
Con il passare del tempo sono subentrati, nell'ambito della DeFi, diversi diretti concorrenti di Ethereum tra cui la tecnologia che farà parte dell'argomento di studio di questa tesi, ovvero Algorand [11]. Algorand è una blockchain concepita nel 2017

da Silvio Micali, professore del MIT di Boston e vincitore nel 2012 del premio Turing attraverso il suo contributo nell'ambito della crittografia.

Figura 1.2. Logo Algorand

Source: https://icoholder.com/it/companies/algorand-11804

La criptovaluta della blockchain di Algorand si chiama Algo. l'Algo beneficia delle peculiarità della blockchain di Algorand quali:

- **Velocità**: Un Algo può essere inviato a chiunque in meno di 4.5 secondi.
- **Scalabilità**: La blockchain di Algorand è stata designata per miliardi di utenti.
- **Fiducia**: Blockchain priva di fork, fenomeno che compromette la validazione di transazioni.
- **Trasparenza**: Il totale di Algo esistente è di 10 miliardi ed è sempre possibile visionare l'ammontare di Algo attualmente in circolazione.
- **Sicurezza**: Attraverso la crittografia viene garantita la protezione di dati personali e informazioni riguardanti pagamenti di Algo.

Algorand nasce con lo scopo principale di abbattere quelle che sono state e che sono ancora ad oggi gli ostacoli principali che minano la diffusione delle tecnologie blockchain all'interno della nostra quotidianità, ovvero decentralizzazione, scalabilità, e sicurezza. Algorand ha creato una blockchain unica che si basa su cinque elementi tecnologici chiave:

- Pure Proof of Stake (PPoS)
- Scalabilità globale
- Decentralizzazione
- Sicurezza
- Finalità garantita

Il Pure Proof of Stake (PPoS) è un protocollo di consenso, ovvero, un insieme di regole che nelle tecnologie a registro distribuito, consentono ai nodi partecipanti alla rete su cui agisce, di ottenere un consenso all'unanimità per la validazione di un nuovo blocco da aggiungere alla catena [6]. I nodi sono i computer o i dispositivi che formano una rete decentralizzata peer-to-peer e permettono la diffusione delle informazioni relative alle transazioni e ai blocchi.
Tra i protocolli di consenso più diffusi troviamo il Proof of Work (PoW), protocollo creato con la nascita di Bitcoin e in seguito utilizzato anche da Ethereum e da molte altre criptovalute. Il processo che caratterizza il PoW è conosciuto come "mining", e di conseguenza, i nodi che ne prendono parte vengono definti "miners". Questo meccanismo richiede che i miners risolvano un complesso enigma crittografico sulle loro macchine. Il primo a riuscirci ottiene il diritto di aggiungere un nuovo blocco alla catena. Questo protocollo di consenso non è sicuramente privo di difetti, infatti, si trova a dover affrontare alcuni problemi come la scalabilità, la vulnerabilità, il forking e l'alto consumo di energia [13]. Proprio per risolvere i problemi relativi ai consumi energetici legati al Proof of Work, è nato il Proof of Stake (PoS), un protocollo di consenso che conferisce l'attribuzione del potere di approvazione in base al numero di monete possedute da un partecipante al processo di consenso. Nonostante le migliorie riscontrate rispetto al PoW, anche i protocolli Proof of Stake sono soggetti a problemi tecnici, spesso dovuti a centralizzazione o sicurezza. [7]. Algorand attraverso il Pure Proof of Stake (PPoS) mira proprio a colmare quelle che sono le lagune e le problematiche che si verificano con l'utilizzo dei protocolli di consenso appena descritti. Il PPoS è un protocollo che ha come obiettivo quello di far si che il potere di validazione non venga conferito ad utenti male intenzionati che potrebbero validare l'aggiunta dei così detti "fake block". Ciò avviene scegliendo in modo totalmente casuale e segreto un certo numero di nodi partecipanti alla rete, i quali avranno la possibilità di proporre nuovi blocchi o votare sulle proposte di blocco che gli vengono sottoposte. Nel capitolo seguente verrà approfondito più nel dettaglio il funzionamento di questo protocollo di consenso [13].

1.2 Presentazione del problema

Come già accennato nella prima parte dell'introduzione, gli smart contract costituiscono il cuore della finanza decentralizzata e di tutte le DApp. Le normali applicazioni web basate su server, ad oggi disponibili, privano l'utente della possibilità di visionare la logica interna dell'applicazione e non consentono di avere il controllo sull'ambiente di esecuzione. Ne consegue che l'utente deve necessariamente fidarsi

del fornitore dei servizi dell'applicazione. Gli smart contract garantiscono agli utenti sicurezza, affidabilità e trasparenza in quanto sfruttano le peculiarità offerte dalla tecnologia blockchain. Attraverso gli smart contract sono implementabili molteplici casi d'uso, tra cui, la possibilità di mantenere criptovalute assumendo un ruolo di custodia e gestire il rilascio di tali risorse in modo totalmente autonomo e personalizzabile. Ciò garantisce un'ampia flessibilità nella creazione di nuove applicazioni decentralizzate.
Colui che ha coniato il termine smart contract è stato Nick Szabo nel 1994 [10]. In seguito il concetto di smart contract è stato ripreso da Vitalik Buterin, fondatore di Ethereum, ad oggi la più grande piattaforma di contratti intelligenti decentralizzata basata su blockchain [18]. La blockchain di Ethereum è stata la prima a comprendere le potenzialità degli smart contract. In questi anni gli smart contract in Ethereum sono stati soggetto di analisi e discussioni, da cui sono emerse alcune problematiche. Prendiamo in analisi un esempio: Supponiamo di voler implementare un'Atomic swap, ovvero una situazione in cui due parti, A e B, possiedono monete e vogliano effettuare uno scambio tra di loro senza dover dare fiducia a terze parti. Il principio alla base è l'atomicità dello scambio, cioè, o avvengono entrambi i trasferimenti, o non ne avviene alcuno. Un'implementazione di tale situazione attraverso l'uso degli smart contract di Ethereum richiederebbe un contratto di tipo "hashed timelock contract", un protocollo molto delicato, multifase, e con una programmazione molto rischiosa. D'altra parte Algorand attraverso i suoi smart contract, implementati on-chain su Layer-1, fornisce una soluzione semplice e sicura a queste problematiche. Un altro aspetto importante che spesso Ethereum è costretto ad affrontare è quello della scalabilità. Si è riscontrato questo problema poiché ogni singola esecuzione di un contratto intelligente di Etheruem causa il blocco dell'intera blockchain, portando inevitabilmente ad un rallentamento della velocità con cui i nuovi blocchi vengono generati. I contratti off-chain di Algorand sono stati concepiti appunto per ovviare ai problemi relativi alla scalabilità. Il loro funzionamento è paragonabile all'uso di un normale conto corrente, dove ogni utente può compilare i propri assegni e i fondi vengono trasferiti una volta che l'assegno viene incassato, il tutto evitando che l'utente si metta in coda in banca. Questo sistema consente ad Algorand di avere grandi benefici in termini di prestazioni [8].

1.3 Sinossi del contributo

In questo elaborato analizzeremo quelle che sono le proprietà che caratterizzano la blockchain di Algorand, e più precisamente studieremo le potenzialità e le possibilità di utilizzo che gli smart contract di Algorand offrono. Questo studio verterà sulla

creazione e la realizzazione di smart contract su piattaforma Algorand applicati ad un particolare caso di studio che verrà ampiamente definito nei capitoli che seguono. Il caso di studio che verrà esposto rientra in quella che viene definita economia circolare, ovvero un particolare modello economico basato sul riuso continuo delle risorse, l'eliminazione degli sprechi, la riparazione, e il riciclaggio. Questo sistema mira a ridurre al minimo la generazione di nuove risorse, favorendo il riutilizzo e il rinnovamento di quelle già esistenti. L'economia circolare cerca di favorire e migliorare la produttività delle risorse che normalmente andrebbero smaltite [19]. Algorand, inoltre, è una tecnologia che ha fatto della sostenibilità uno dei suoi obiettivi primari, di fatti, garantisce altissime prestazioni pur mantenendo i consumi in termini di elettricità paragonabili a quelli di un normale server per applicazioni centralizzate.

1.4 Struttura dell'elaborato

Lo studio realizzato è strutturato su 6 capitoli nei quali verranno illustrate le funzionalità e le caratteristiche di Algorand e dei suoi smart contract attraverso la realizzazione di un'applicazione pratica al fine di mostrare l'uso di questa tecnologia da un punto di vista concreto.
Nel primo capitolo è stata esposta un'introduzione riguardante l'ambito in cui Algorand lavora e le differenze che i suoi smart contract possiedono rispetto a quelli offerti da tecnologie più conosciute come Ethereum. Nel capitolo 2 verranno descritte tutte le componenti che Algorand sfrutta e che consentiranno di comprendere a pieno l'utilizzo degli smart contract nel caso di studio illustrato in seguito. Nel capitolo 3 verranno analizzati in modo dettagliato tutte le tipologie di smart contract di cui Algorand consente l'utilizzo e le loro specifiche caratteristiche. Nel capitolo 4 troveremo una descrizione dettagliata del caso di studio, mentre nel capitolo successivo, verrà illustrata l'implementazione di quest'ultimo e la relativa valutazione. Infine nel capitolo 6 verranno esposte le conclusioni relative allo studio realizzato e un'analisi dei possibili sviluppi futuri.

Capitolo 2

La Piattaforma Algorand

In questo capitolo verranno mostrate tutte le componenti tecnologiche che Algorand offre. Queste permetteranno di comprendere a pieno quali sono le possibili modalità di utilizzo degli smart contract. Inizialmente verrà fornita con una descrizione riguardante la tecnologia blockchain e più in generale verrà data una definizione di DLT. In seguito verrà esposto il così detto "blockchain Trilemma" e cosa fa Algorand per risolverlo. Successivamente verranno descritte le caratteristiche alla base della blockchain di Algorand come, nodi di rete, account, wallet e tipologie di transazioni. Infine verranno introdotti gli elementi che compongono il core protocol di Algorand, ovvero, Atomic transfers e Algorand Standard Asset, esponendo le loro caratteristiche e il loro funzionamento.

2.1 Blockchain e DLT

Le DLT o tecnologie a registro distribuito sono una sorta di database che consente di mantenere informazioni in modo condiviso e duplicato su una rete formata da computer che prendono il nome di nodi. Il registro è accessibile da parte di tutti i nodi e inoltre le informazioni in esso contenute sono sempre verificate. La blockchain è un particolare tipo di tecnologia a registro distribuito.
La blockchain è un meccanismo che utilizza la crittografia e dei particolari algoritmi matematici per la creazione e la gestione di una struttura dati che prende la forma di una catena composta da blocchi. In questa particolare struttura dati le informazioni che vengono inserite sono irremovibili, distribuite e pubbliche.
La blockchain inoltre è una tecnologia che può prendere diverse forme, quali:

- **Blockchain aperta e senza autorizzazione**: In questo tipo di blockchain un utente può prendere parte alla rete di nodi e abbandonarla successivamente in modo totalmente libero, senza quindi la necessità di essere autorizzato da

alcuna entità centrale. Questo tipo di blockchain non possiede un proprietario centrale della rete e le informazioni mantenute nel registro sono distribuite a tutti i nodi che la compongono.

- **Blockchain con autorizzazione**: Le blockchain con autorizzazione, a differenza delle precedenti, prevedono la presenza di un'amministratore, il quale ha il compito di imporre le regole che consentono l'accesso alla reti di nodi. Questo metodo ha il vantaggio di poter verificare con facilità l'identità dei partecipanti. D'altro canto gli stessi partecipanti sono obbligati a dare fiducia all'amministratore di rete, e ciò risulta essere un notevole svantaggio. Le blockchain che richiedono autorizzazione possono essere ulteriormente suddivise in due categorie, ovvero, blockchain pubbliche con autorizzazione e blockchain con autorizzazione aziendale. Nelle prime l'accesso e la visione della rete è consentita a chiunque, ma solo chi ha superato una fase di autorizzazione ha la facoltà di generare transazioni o alterare lo stato del ledger. Nelle blockchain con autorizzazione aziendale invece, l'accesso ha delle restrizioni e l'unico che può alterare lo stato del ledger e generare transazioni è l'amministratore di rete.

In entrambe le tipologie di blockchain descritte le transazioni che vengono generate non necessitano dell'intermediazione di terze parti.
Quando un nuovo utente prende parte alla rete di nodi, questo viene fornito di due chiavi, una pubblica, che rappresenta l'indirizzo dell'utente sulla rete, e una privata, il cui scopo è quello di creare una firma digitale per le transazioni. Entrambe le chiavi vengono memorizzate all'interno di un wallet digitale. Quando un utente genera una transazione diretta verso un altro utente questa verrà firmata con la chiave privata del mittente e verrà inviata verso un secondo indirizzo, ovvero, la chiave pubblica del destinatario. Successivamente a questo passaggio la transazione viene inserita all'interno di un blocco che verrà sottoposto ad un processo di validazione, cioè quel processo in cui la transazione deve essere approvata dai partecipanti della rete. Se tale verifica viene superata con successo la transazione viene trasferita e il nuovo blocco viene aggiunto alla catena [14].

2.1.1 Blockchain Trilemma

La creazione di piattaforme blockchain si basa su tre aspetti principali, decentralizzazione, scalabilità, e sicurezza. È proprio su questi concetti che nasce il cosiddetto "blockchain trilemma", illustrato in Fig. 2.1. Il blockchain trilemma afferma che se si vuole creare una DLT non è possibile sostenere la presenza di tutti e tre i

concetti sopra definiti contemporaneamente, bensì solo due di questi [13]. Da questa affermazione emergono tre casi differenti:

- Senza **decentralizzazione**: Un sistema senza decentralizzazione risulta essere esclusivo e segreto, molto simile ai sistemi ad oggi diffusi basati su autorità centrali.
- Senza **sicurezza**: Un sistema privo di sicurezza non garantisce la trasparenza sulle transazioni che vengono generate, ciò implica che potremmo essere ingannati ricevendo transazioni contraffatte.
- Senza **scalabilità**: Un sistema non scalabile non è in grado di far fronte all'aumentare di utenti e transazioni, ciò porta inevitabilmente ad un rallentamento del sistema e a possibili congestioni di rete.

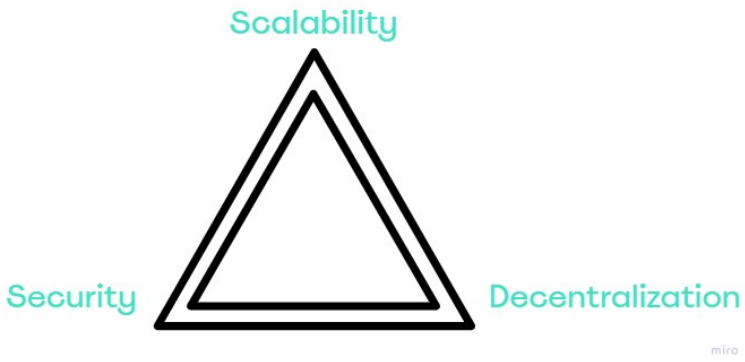

Figura 2.1. Blockchain Trilemma

2.1.2 Pure Proof of Stake (PPoS)

Per cercare di risolvere il trilemma sono stati sviluppati diversi protocolli di consenso, ovvero, insiemi di regole dedite all'approvazione di nuovi blocchi, con l'obiettivo di garantire scalabilità, sicurezza e decentralizzazione. Tra tutti i protocolli sviluppati, Algorand ha ideato quello che più di tutti è in grado di garantire questi tre aspetti, il Pure Proof of Stake (PPoS). Questo meccanismo di consenso prevede che per ogni round di approvazione vengano selezionati un certo numero di account per proporre nuovi blocchi, e ciò avviene attraverso una funzione chiamata VRF, acronimo di Variable Random Function. La VRF è uno strumento di randomizzazione crittografica che ricevuto in input una chiave segreta e un valore, produce un output pseudocasuale, contenente una prova che chiunque può utilizzare per verificare il risultato. Questa funzione viene utilizzata per scegliere i leader che proporranno i nuovi blocchi e i membri del comitato di consenso che avranno il compito di votare

le proposte di blocco. Il suo funzionamento è simile a quello di una lotteria. Questa funzione viene eseguita per ogni singolo account online presente sulla rete. Maggiore è la quantità di Algo in possesso di un account, ovvero il suo "stake", e maggiore sarà la sua probabilità di essere selezionato. Per evitare problemi di esposizione, l'utente che partecipa al consenso genera una chiave di partecipazione, valida per un determinato numero di round. Il processo di consenso è composto da tre fasi distinte:

- **Block Proposal**: Durante questa fase vengono selezionati gli account per la proposta di nuovi blocchi. Viene eseguita la funzione VRF su ogni account online in possesso chiavi di partecipazione valide. Qualora un account venga selezionato, il nodo in cui è contenuto genera una proposta di blocco e la propaga all'interno della rete insieme all'output della funzione VRF, il quale ne prova la validità.

- **Soft Vote**: L'obiettivo di questa fase è quello di garantire che un solo blocco venga certificato. In questa fase ogni nodo riceve numerosi messaggi di proposta, verifica la firma del messaggio e in seguito convalida la selezione utilizzando la prova VRF. Per concludere, il nodo effettuerà una comparazione tra i diversi hash delle prove VRF in modo da determinare e propagare nella rete solo la proposta di blocco con hash minore. A questo punto la funzione VRF viene eseguita nuovamente su tutti i nodi che hanno account con chiavi di partecipazione valide al fine di votare sulla proposta di blocco. Se un account viene selezionato questo propagherà il suo voto all'interno della rete, ponderato in base al numero di Algo in suo possesso. Ogni nodo accumula i voti in circolazione sulla rete fino al raggiungimento di una maggioranza e, una volta raggiunta, la proposta di blocco passa alla fase finale.

- **Certify Vote**: Questa è la fase finale del consenso. Similmente alle fasi precedenti viene scelto un nuovo comitato con il compito di certificare la proposta di blocco ricevuta nella fase di Soft Vote. I voti vengono convalidati da ogni nodo fino al raggiungimento di una maggioranza, e una volta raggiunta, viene richiesto al nodo di creare un certificato per il nuovo blocco che verrà finalmente approvato e aggiunto alla catena.

Una volta terminate queste tre fasi inizia un nuovo round e il processo si ripete nuovamente.
Per concludere la descrizione di questo protocollo possiamo affermare che il Pure Proof of Stake garantisce:

- **Scalabilità**: L'esecuzione del PPoS può essere messa in atto anche da un comune laptop dato che non necessita di un'alta potenza computazionale, ciò lo rende accessibile ad un vasto numero di utenti.
- **Decentralizzazione**: Lo stake espresso in termini di Algo ha lo stesso potere di quello di ogni altro utente, ciò garantisce l'assenza di entità centrali in grado di esercitare maggiore potere sugli altri partecipanti alla rete.
- **Sicurezza**: Il PPoS genera ad ogni round il nuovo comitato in modo casuale e privato eliminando qualsiasi comunicazione con autorità centrali. Questo previene eventuali attacchi di terze parti, data la mancanza di informazioni da intercettare. Inoltre quando la prova di vincita e la decisione sul blocco vengono propagate all'interno della rete, e quindi rese pubbliche, un tentativo di corruzione del comitato risulterebbe inutile in quanto la decisione è stata già presa e diffusa. [6].

2.1.3 Nodi Algorand

Nel paragrafo precedente si è parlato di nodi di rete. La rete di Algorand prevede due tipi di nodi, i quali hanno compiti e caratteristiche diverse:

- **Nodo di inoltro**: Sono definiti come nodi di comunicazione e fungono da fulcri della rete. Questi nodi consentono di avvantaggiare la propagazione di messaggi di protocollo inviati dai nodi di partecipazione, grazie a connessioni di rete ad alte prestazioni. Questi nodi hanno il compito di controllate le firme dei messaggi che ricevono, e successivamente propagare quelli che sono stati ritenuti validi.
- **Nodi di partecipazione**: Sono i nodi che prendono attivamente parte al processo di consenso e possono rappresentare più utenti, a patto che vengano installate le corrette chiavi di partecipazione. Questi blocchi hanno il compito principale di proporre e votare sulle proposte di blocco. La comunicazione tra questi tipi di nodo avviene grazie ai nodi di inoltro [6].

2.2 Wallet e Account

Algorand definisce gli account come coppie di chiavi, una pubblica e una privata. Queste vengono generate attraverso l'uso di un valore casuale sottoposto ad una libreria, la quale restituisce in output 2 array, della lunghezza di 32 byte ciascuno, rappresentanti appunto le chiavi appena citate. Il processo appena descritto è illustrato nella Fig. 2.2.

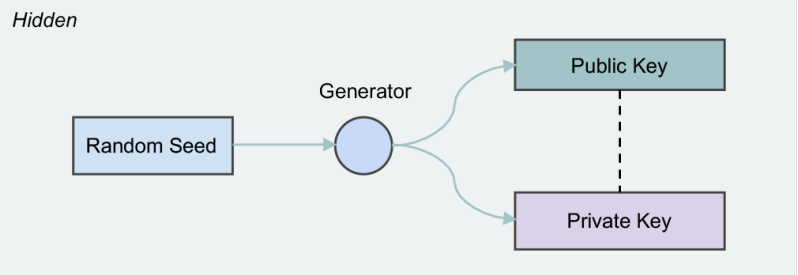

Figura 2.2. Generazione chiave Pubblica/Privata

Source: https://developer.algorand.org/docs/features/accounts/

Per rendere le chiavi più leggibili,e facilitarne quindi l'utilizzo da parte degli utenti, queste subiscono modifiche e trasformazioni. Di fatti la chiave pubblica viene trasformata attraverso un processo nel quale viene aggiunto all'array di 32 byte, una checksum della lunghezza di 4-byte. In seguito per concludere questa trasformazione il nuovo array viene codificato in base 32 (Fig. 2.3). Ciò che si ottiene da questi passaggi è un indirizzo Algorand.

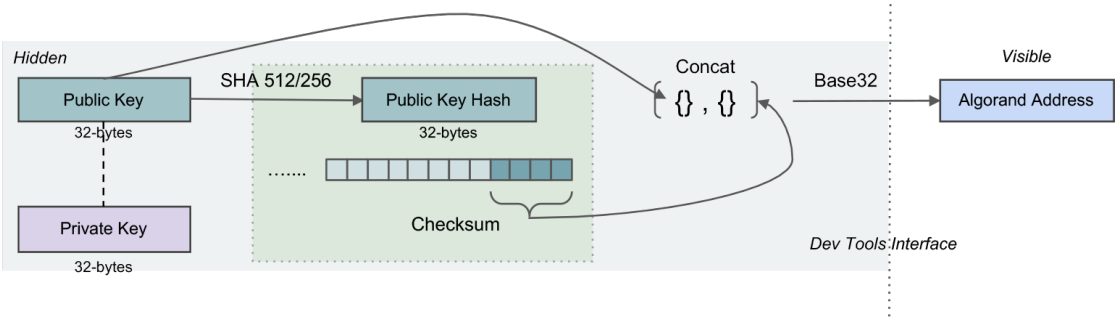

Figura 2.3. Da chiave pubblica a indirizzo Algorand

Source: https://developer.algorand.org/docs/features/accounts/

Anche la chiave privata subisce trasformazioni, infatti viene convertita in una lista di 25 parole che ne migliora l'usabilità da parte dell'utente. Questa trasformazione avviene convertendo la chiave privata in numeri interi a 11 bit e poi creando una corrispondenza tra questi numeri e una lista di parole (l'intero n corrisponde all'n-esima parola). Questo processo genera una lista di 24 parole. In seguito i primi due byte dell'hash della chiave privata subiscono la stessa procedura e si ottiene una singola parola che aggiunta alla lista precedente forma la chiave privata mnemonica, ovvero, una lista di 25 parole. (Fig. 2.4).

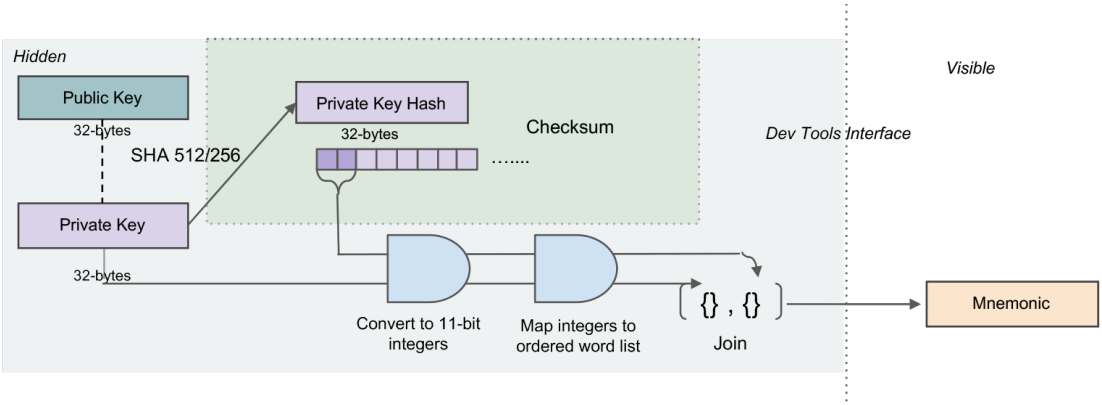

Figura 2.4. Chiave Privata Mnemonica

Source: https://developer.algorand.org/docs/features/accounts/

Gli account in Algorand sono identificati attraverso l'indirizzo generato dalla trasformazione della chiave pubblica e ad essi vengono associati altri due attributi memorizzati sulla blockchain di Algorand, ovvero, "balance", che rappresenta il saldo in termini di Algo dell'account, e "status", che assume un valore booleano true o false per indicare se l'account in questione partecipa o meno alla protocollo di consenso. Inoltre un account per essere considerato attivo e poter inviare transazioni richiede che sia in possesso di un saldo minimo pari a 0.1 Algo (100000 microAlgo). Algorand fornisce anche la possibilità di creare degli account multifirma, cioè una rappresentazione logica di un insieme ordinato di indirizzi associati ad un valore di threshold e un valore di versione. Questi particolari account possono eseguire le stesse operazioni di un normale account e vengono utilizzati per avere un maggiore livello di sicurezza richiedendo una firma multipla per autorizzare le transazioni da essi generate
Per concludere questo paragrafo definiamo quelli che sono i wallet in Algorand, ovvero, strumenti di sviluppo generati e gestiti da un processo chiamato Key Management Daemon (kmd) con l'obiettivo di memorizzare collezioni di chiavi. Ad ogni wallet viene associata una chiave master, rappresentata come una chiave mnemonica di 25 parole, derivante dalla collezione di chiavi che il wallet mantiene [4].

2.3 Transazioni

In Algorand le transazioni hanno dei campi comuni, e dei campi specifici per ogni tipologia di transazioni. I campi comuni a tutti i tipi di transazione sono i seguenti:

- **Fee**: Tariffa di commissione della transazione.
- **FirstValid**: Round da cui la transazione sarà valida.

- **GenesisHash**: Hash del blocco genesi relativo al round FirstValid.
- **LastValid**: Round dopo il quale la transazione non è più valida.
- **Sender**: Indirizzo del mittente della transazione.
- **TxType**: Tipo della transazione.
- **GenesisID**: Rete della transazione.
- **Group**: Hash del gruppo di transazioni se ne fa parte.
- **Lease**: Implementa la mutua esclusione
- **Note**: Qualsiasi tipo di dato fino a 1000 byte.
- **RekeyTo**: Specifica l'indirizzo autorizzato.

Algorand offre 5 tipologie diverse di transazione e prevede anche la possibilità di generare dei sottotipi di transazione per effettuare operazioni particolari. Ognuna di questi sottotipi di transazioni richiede la conoscenza della corretta combinazione di campi da specificare per ottenere la transazione che svolga l'operazione desiderata. I tipi di transazione previsti dal protocollo Algorand sono i seguenti:

1. **Payment**: Viene identificata dalla parola `pay`. Questo tipo di transazione è sicuramente la più comune e viene utilizzata per inviare un determinato quantitativo di Algo da un account ad un altro. Attraverso una transazione di questo tipo, specificando un particolare campo chiamato `CloseRemainderTo`, è possibile eliminare un account dal ledger di Algorand. Un'esempio della struttura di questa tipologia di transazioni è illustrata nella Fig. 2.5.

2. **Key Registration**: Viene identificata dalla parola `keyreg`. Questa tipologia di transazione ha lo scopo di registrare un nuovo account sulla rete di Algorand. Una transazione di tipo Key Registration può essere relativa ad un account offline, nel caso in cui l'account non partecipa al protocollo di consenso, o online, nel caso in cui vi partecipi il che richiede la specifica di particolari campi.

3. **Asset Configuration**: Viene identificata dalla parola `acfg`. Questa tipologia di transazione riguarda l'uso di una delle componenti del core protocol di Algorand, ovvero, gli Algorand Standard Asset che saranno approfonditi nei paragrafi successivi. Le transazioni di tipo Asset Configuration hanno tre scopi differenti, e vale la pena descriverli:

- Creazione di un Asset: Transazione utilizzata per la creazione di un nuovo asset, si distingue dal fatto che nella transazione non vi è alcuna specifica nel campo relativo all'id dell'asset mentre sono presenti i parametri relativi alla sua struttura.
- Riconfigurazione di un Asset: Transazione che viene utilizzata per effettuare modifiche sui parametri di struttura dell'asset. Ciò che differenzia questo tipo di transazione dal precedente è la presenza di un valore nel campo relativo all'id.
- Eliminazione di un Asset: Transazione utilizzata per l'eliminazione definitiva di un asset dalla rete Algorand. Questa transazione è possibile se solo se tutte le unità dell'asset sono in possesso del creatore e si differenzia dalle transazioni precedenti per la presenza dell'id dell'asset e la mancanza dei parametri di struttura.

4. **Asset Freeze**: Viene identificata dalla parola `afrz`. Questo tipo di transazione è utilizzata per far si che un indirizzo non sia più in grado di inviare o ricevere l'asset specificato all'interno della transazione.

5. **Asset Transfer**: Viene identificata dalla parola `axfer` e i suoi utilizzi sono i seguenti:

   - Opt-in di un asset: Transazione utilizzata per aggiungere un'asset al proprio wallet in modo da abilitare la possibilità di riceverne e inviarne delle unità. Questo tipo di transazione è caratterizzata dal fatto che sia il mittente che il destinatario corrispondono allo stesso indirizzo.
   - Trasferimento di un asset: Transazione utilizzata per l'invio di asset da un mittente verso un destinatario. Il suo schema generale è simile a quello di una Payment Transaction.
   - Revoca di un Asset: Transazione utilizzata per revocare un certo quantitativo di unità di un Asset da uno specifico indirizzo. Questa particolare transazione richiede che l'indirizzo relativo all'account che subisce la revoca venga specificato all'interno di un campo chiamato `AssetSender`, e che il mittente della transazione sia l'indirizzo di "Clawback", definito precedentemente mediante una transazione di tipo Asset Configuration [4].

```
{
  "txn": {
    "amt": 5000000,
    "fee": 1000,
    "fv": 6000000,
    "gen": "mainnet-v1.0",
    "gh": "wGHE2Pwdvd7S12BL5FaOP20EGYesN73ktiC1qzkkit8=",
    "lv": 6001000,
    "note": "SGVsbG8gV29ybGQ=",
    "rcv": "GD64YIY3TWGDMCNPP553DZPPR6LDUSFQOIJVFDPPXWEG3FVOJCCDBBHU5A",
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
    "type": "pay"
  }
}
```


Figura 2.5. Esempio di transazione di tipo Payment

Source: https://developer.algorand.org/docs/features/transactions/

2.3.1 Autorizzazione

Qualsiasi transazione generata necessita anche di essere autorizzata prima dell'invio alla rete. L'autorizzazione è un processo che deve essere compiuto dal mittente della transazione, ovvero, l'indirizzo specificato nel campo `Sender`. La fase di autorizzazione consiste nell'aggiunta di una firma all'oggetto `Txn` [4]. In seguito a questa fase l'oggetto viene inserito a sua volta all'interno di un nuovo oggetto di tipo `SignedTxn`, il quale contiene la transazione firmata e il tipo di firma che ha ricevuto. Esistono diversi tipi di firma che consentono l'autorizzazione di una transazione:

- **Firma Singola**: È una firma risultante dalla chiave privata del mittente della transazione. La parola che identifica questo tipo di firma è `sig`. Nella Fig. 2.6 viene mostrata una transazione di tipo Payment a seguito del processo di autorizzazione.
- **Firma Multipla**: È la firma che viene utilizzata quando il mittente della transazione è l'indirizzo di un account multiplo. Questo tipo di firma richiede che il numero di indirizzi che la effettuano siano maggiori o uguali al valore di threshold. Questa tipologia di firma viene identificata dalla parola `msig`.
- **Firma Logica**: È la tipologia di firma che viene utilizzata quando il mittente di una transazione è un Algorand Smart Contract, più precisamente un contratto stateless. Questo tipo di firma sarà argomento del prossimo capitolo dove verrà esposta nel dettaglio.

```
{
  "sig": "ynA5Hmq+qtMhRVx63pTO2RpDrYiY1wzF/9Rnnlms6NvEQ1ezJI/Ir9nPAT6+u+K8BQ32pplVr
  "txn": {
    "amt": 10000000,
    "fee": 1000,
    "fv": 4694301,
    "gen": "testnet-v1.0",
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 4695301,
    "rcv": "QC7XT7QU7X6IHNRJZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HSVQ",
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
    "type": "pay"
  }
}
```


Figura 2.6. Esempio di transazione di tipo Payment autorizzata con firma singola

Source: https://developer.algorand.org/docs/features/transactions/signatures//

2.4 Asset

Algorand garantisce la possibilità di creare quelli che vengono chiamati Algorand Standard Asset (ASA), caratteristica offerta dal layer-1 di Algorand. Questa funzionalità consente agli utenti di rappresentare risorse, come ad esempio stablecoin, punti fedeltà o crediti di sistema, pur mantenendo la stessa velocità e facilità d'uso dell'Algo nativo. Gli Asset sono impiegabili in moltissimi modi differenti ed è possibile applicare ad essi particolari restrizioni in modo da adattarli al caso d'uso che necessitiamo di rappresentare [15]. Gli ASA consentono di mettere in pratica il Role Based Asset Control (RBAC), ovvero un controllo opzionale e flessibile degli asset per gestire requisiti aziendali e normativi. Tra le possibilità disponibili troviamo le seguenti:

- Forzare il trasferimento di asset nel caso in cui le normative lo richiedano.
- Creazione di liste con privilegi. Ciò consente solo a specifici indirizzi di utlizzare un asset, ponendo quindi delle limitazioni ad altri account.
- Mantenimento di risorse non coniate su indirizzi di riserva per creare requisiti aziendali personalizzati.
- Inclusione on-chain, nella definizione dell'asset, di collegamenti relativi alla documentazione off-chain dell'asset [4].

Uno dei progetti che sta avendo maggior successo e che utilizza questo tipo di risorse è PlanetWatch, compagnia con cui Algorand collabora. PlanetWatch offre un sistema che prevede ricompense sotto forma di Algorand Standard Asset nominati PLANET, per chi invia dati validi riguardo il livello d'inquinamento dell'aria attraverso sensori

IoT, cioè sensori che consentono di convertire fenomeni fisici in dati digitali [16].
Ci sono alcune nozioni che bisogna conoscere per comprendere a pieno l'utilizzo degli Algorand Standard Asset. Ad un singolo utente è consentito creare fino ad un massimo di 1000 ASA differenti. Inoltre per ogni Asset che si crea o che si possiede viene richiesto che il saldo minimo dell'account venga incrementato di 0.1 Algo, equivalentemente a quanto accade per la moneta nativa. Un'ultima nozione molto importante da conoscere è quella dell'Opt-in. Prima che un account possa ricevere un determinato asset, è necessario che l'utente invii una transazione di tipo Opt-in dal suo wallet, specificando l'id dell'asset, ponendosi sia come mittente che come destinatario della transazione, e assegnando il valore 0 al campo relativo all'ammontare di unità dell'asset. Questa operazione consentirà al wallet di poter inviare e ricevere unità dell'asset specificato.
Al momento della creazione di un ASA è possibile configurare alcuni parametri, che sono i seguenti:

- **Parametri Immutabili**:
  - Creator (obbligatorio)
  - AssetName (opzionale, ma raccomandato)
  - UnitName (opzionale, ma raccomandato)
  - Total (obbligatorio)
  - Decimals (obbligatorio)
  - DefaultFrozen (obbligatorio)
  - URL (opzionale)
  - MetaDataHash (opzionale)
- **Parametri Mutabili**:
  - **Manager Address**: Indirizzo relativo all'unico account a cui è consentito eliminare un asset e riconfigurare gli altri parametri mutabili. Questo indirizzo ha anche la possibilità di privarsi di questo ruolo e assegnare il privilegio di manager ad un altro account.
  - **Reserve Address**: Indirizzo relativo all'account nel quale vengono riposte le unità di asset non coniate. Il suo scopo è quello di mantenere delle disponibilità per possibili usi futuri. Quando le unità di asset vengono trasferite da questo account diventano unità coniate e utilizzabili a tutti gli effetti.
  - **Freeze Address**: Indirizzo che può effettuare il congelamento delle disponibilità di asset di un account. Un account che viene bloccato da

una transazione di tipo Asset Freeze, viene privato della possibilità di inviare e ricevere unità dell'asset in questione. Questa funzionalità è stata resa disponibile per porre restrizioni sulle modalità di utilizzo dell'asset, in modo tale da venire incontro alle esigenze del caso d'uso in cui viene impiegato.

– **Clawback Address**: Indirizzo relativo all'account che può inviare transazioni di revoca. Questo ruolo consente di privare un determinato account delle disponibilità di asset che mantiene. Una transazione di revoca potrebbe risultare necessaria qualora un account non rispetti determinate regole contrattuali relative al campo di utilizzo dell'asset, o più semplicemente nel caso in cui si vuole eliminare l'asset dalla blockchain di Algorand e quindi c'è la necessità che tutte le unità coniate tornino in mano al Creator Address. Per inviare transazioni di revoca è necessario specificare un campo chiamato AssetSender, relativo all'account che verrà privato delle sua disponibilità. [4].

Nella Fig. 2.7 vengono descritte le diverse tipologie di transazioni che coinvolgono gli Algorand Standard Asset con i relativi campi di transazione che necessitano di essere definiti per generarle.

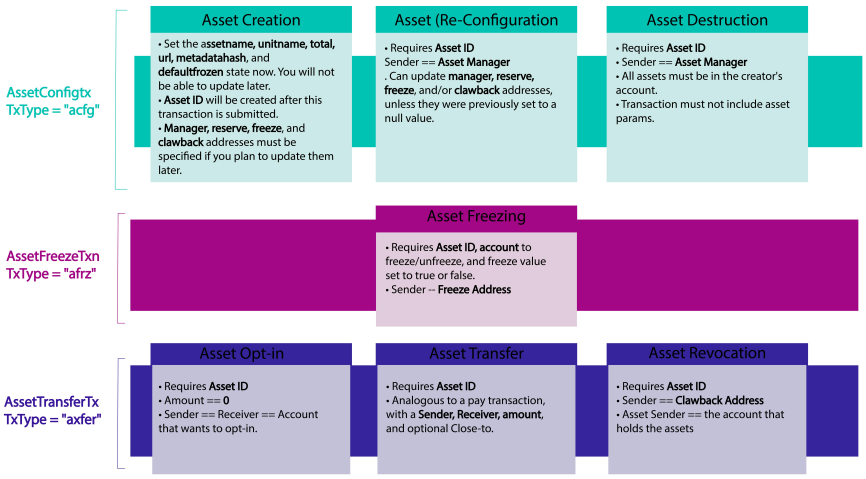

Figura 2.7. Schema esemplificativo per la creazione di transazioni relative ad ASA

Source: https://medium.com/algorand/algorand-standard-assets-efda8afcfc0a

2.5 Atomic Transfer

Tra le più interessanti funzionalità che Algorand racchiude all'interno del suo Layer-1 troviamo gli Atomic Transfer, ovvero, il raggruppamento di più transazioni in un unico gruppo. Questa funzionalità è nata con l'obiettivo di eliminare i problemi relativi alla sfiducia tra le parti coinvolte nelle transazioni. Infatti nella finanza tradizionale quando due parti vogliono effettuare uno scambio di risorse, viene spesso richiesta la presenza di un intermediario che garantisca la corretta finalità dello scambio. Algorand offre come soluzione a questa problematica gli Atomic Transfer i quali consentono di effettuare scambi di risorse tra due sconosciuti garantendo sicurezza e affidabilità. Questa tipologia di trasferimenti gode ovviamente dell'atomicità, questo significa che le transazioni sono indivisibili in termini di finalità. Questi trasferimenti vengono implementati come operazioni batch irriducibili, eliminando la necessita di creare soluzioni complesse come succede in Ethereum con gli "Hashed timelock contract". Inoltre i trasferimenti atomici vengono inviati e confermati al di sotto di 5 secondi, esattamente al pari di una normale transazione e consentono il trasferimento sia di Algo che di ASA, oltre ad avere ovviamente la possibilità di essere gestiti attraverso gli smart contract.
Qui di seguito sono elencati alcuni dei più interessanti casi d'uso implementabili attraverso i trasferimenti atomici:

- **Scambi Circolari**: Situazione in cui A paga B se e solo se B paga C e B paga C se e solo se C paga A.
- **Pagamenti Raggruppati**: I pagamenti vengono effettuati tutti contemporaneamente oppure non ne viene effettuato alcuno.
- **Scambi Decentralizzati**: Pagamenti multipli verso destinatari differenti effettuati contemporaneamente [4].

Analizziamo più dettagliatamente il funzionamento degli Atomic Transfer. In primo luogo vengono generate tutte le transazioni che faranno parte del trasferimento atomico e in seguito vengono combinate in un singolo gruppo a cui verrà assegnato uno specifico ID. Una volta che le transazioni sono state raggruppate queste possono essere divise e distribuite tra i vari mittenti che avranno l'autorità di firmarle e autorizzarle (Fig. 2.8). Infine tutte le transazioni vengono sottomesse contemporaneamente alla rete.

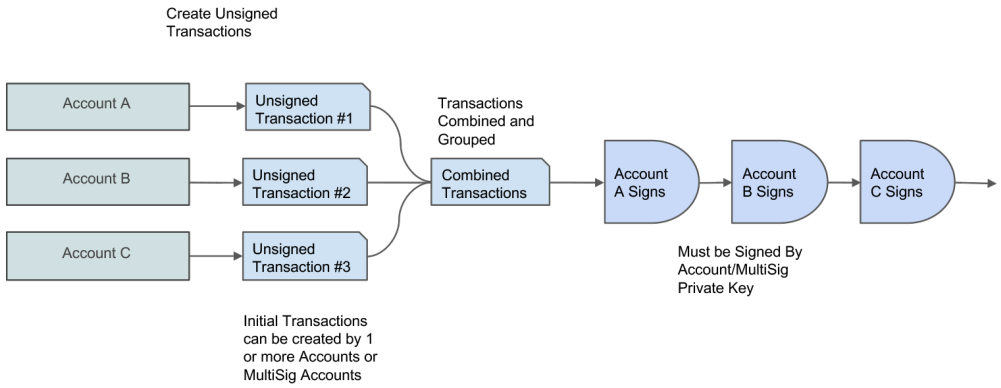

Figura 2.8. Flusso di esecuzione di un Trasferimento Atomico

Source: https://developer.algorand.org/docs/features/atomic_transfers/

Capitolo 3

Smart Contract

In questo capitolo verrà esposto l'argomento cardine dello studio realizzato, gli Algorand Smart Contract (ASC). In primo luogo verrà illustrata una panoramica generale degli smart contract in Algorand evidenziando le funzionalità offerte e le varie possibilità di utilizzo. Successivamente verranno analizzati nel dettaglio i diversi tipi di smart contract implementabili con la blockchain di Algorand, contratti Stateless e contratti Stateful. Infine verrà illustrato il linguaggio TEAL, linguaggio del quale si compongono appunto gli smart contract.

3.1 Introduzione

La maggior parte delle transazioni coinvolte in accordi contrattuali riscontrano spesso problemi legati al dover riporre fiducia a terze parti, o all'alta presenza di pratiche burocratiche. I problemi legati a questo argomento sono il motivo principale per cui si sta diffondendo sempre di più l'utilizzo degli smart contract basati su tecnologia blockchain per eseguire in modo autonomo transazioni e garantire il mantenimento di clausole contrattuali tra le parti coinvolte negli accordi. Nonostante i numerosi vantaggi apportati dagli smart contract è impossibile garantirne la diffusione a causa dei problemi riscontrati dagli sviluppatori che utilizzano blockchain di prima generazione. Alcuni dei problemi che è facile riscontrare sono legati a costi elevati, bassa scalabilità e bassa capacità di adattarsi ai casi d'uso del mondo reale. Se si vogliono realizzare applicazioni realmente efficienti e portare una svolta nel mondo della tecnologia blockchain c'è la necessità di ovviare a questo tipo di problemi. Algorand ha sviluppato degli smart contract che hanno proprio l'obiettivo di abbattere queste barriere. Gli smart contract di Algorand consentono di sviluppare applicazioni complesse in maniera semplice, economica, scalabile e veloce. Questi contratti garantiscono affidabilità essendo eseguiti on-chain, quindi con la certezza che le esecuzioni vadano a buon fine senza subire manomissioni.

Gli smart contract sono uno dei punti cardine del core protocol di Algorand operanti sul Layer-1, e sono divisi in due categorie, contratti con stato e contratti senza stato. La tipologia di contratto determina quando e come verrà valutata la logica del programma. Questi particolari smart contract sono scritti attraverso il linguaggio TEAL (Transaction Execution Approval Language), che verrà esposto dettagliatamente nei paragrafi successivi.
Gli Algorand Smart Contract garantiscono diverse funzionalità e benefici, come:

- **Stateless & Stateful**: Implementazione di contratti senza stato per l'approvazione di transazioni, e implementazione di contratti con stato per la realizzazione di applicazioni complesse.
- **Scalabilità, velocità e sicurezza**: Gli ASC consentono di eseguire oltre 1000 transazioni al secondo, con validazione entro 5 secondi e con assenza di fork.
- **Rischi ridotti**: Grazie ad esecuzioni che non necessitano di fiducia.
- **Bassi costi di esecuzione**: Le transazioni relative agli ASC hanno gli stessi costi di commissione di qualsiasi altra transazione.
- **Facilità d'ingresso nello sviluppo**: Algorand rende lo sviluppo e l'apprendimento relativo agli smart contract estremamente semplice grazie ad una vasta documentazione, alla presenza di template per smart contract Stateless, e di modelli per la creazione di dApp complesse con smart contract Stateful.
- **Implementazione flessibile**: Gli smart contract possono essere applicati a transazioni specifiche o gruppi di transazioni attraverso l'uso degli Atomic Transfer [3].

Algorand è nata con l'obiettivo di garantire alla finanza decentralizzata una blockchain che ne permetta una diffusione ampia e senza impedimenti. Gli Algorand Smart Contract sono nati con questo scopo e svolgono un ruolo chiave in diversi contesti nell'ambito della DeFi. Qui di seguito sono elencati e descritte alcune possibilità di utilizzo degli ASC ai fini di garantire una finanza decentralizzata:

- **Deposito a garanzia (Escrow)**: Considerando un accordo relativo allo scambio di risorse tra due parti, un deposito a garanzia è un conto messo a disposizione da una terza entità in cui vengono riposte le risorse coinvolte nell'accordo. Queste risorse verranno rilasciate alle due parti coinvolte solo nel momento in cui ognuno avrà rispettato i vincoli imposti dall'accordo e al verificarsi di determinate condizioni. Nella finanza tradizionale il ruolo di garante è spesso offerto da intermediari centralizzati come una banca.

In Algorand è possibile implementare questa situazione in modo semplice attraverso l'uso di un contratto stateless in cui verranno codificate le condizioni per il rilascio delle risorse. Il contratto assumerà il ruolo di garante, dato che le risorse verranno depositate sul suo conto e potranno essere rilasciate attraverso l'uso di un trasferimento atomico nel momento in cui le condizioni descritte dalla logica del contratto stesso lo consentano.

- **Stablecoin**: Uno dei fattori principali che mina l'adozione e la diffusione delle criptovalute è l'alta volatilità del prezzo di quest'ultime, poiché le persone tendono ad esitare nell'investire in monete digitali che non hanno un valore accettato come standard. Per eliminare questo problema sono state introdotte quelle che vengono chiamate stablecoins, ovvero, monete digitali che, a differenza delle comuni criptovalute, mantengono un prezzo stabile poiché sono vincolate ad una moneta fiat, ovvero una moneta legale (Come il dollaro o l'euro). Questo vincolo tra stablecoin e monete fiat è reso possibile proprio dall'utilizzo di smart contract. Ciò consente alle persone di acquisire monete digitali senza preoccuparsi di rischi dovuti alle variazioni di prezzo. In Algorand sono già in circolazione stablecoin come Tether e USDC, le quali consentono la creazione di applicazioni veloci e scalabili su blockchain ma con rischi notevolmente ridotti.

- **Credito e prestiti**: Gli smart contract di Algorand consentono di dare maggiori opportunità anche nel settore dei prestiti. Infatti risulta spesso difficoltoso, per chi richiede un prestito, accedere ai servizi finanziari tradizionali. Attraverso gli ASC le organizzazioni che emettono prestiti possono tradurre i vincoli contrattuali in codice informatico al fine di far rispettare le clausole dell'accordo con i mutuatari. Ciò consente di avere costi notevolmente più bassi, poiché viene meno il ruolo di intermediari che nella finanza tradizionale hanno lo scopo di far rispettare i termini dell'accordo. Con questa modalità è possibile offrire servizi di prestito con costi di commissione notevolmente più bassi e di conseguenza renderli accessibili a più persone.

- **Scambi e Liquidità**: Attraverso gli smart contract è possibile scambiare valute digitali in pochissimi secondi. Gli smart contract impiegati in questo tipo di scambi decentralizzati sono in grado verificare e finalizzaere automaticamente le transazioni senza la necessità di coinvolgere alcun intermediario.

- **Soluzioni di pagamento**: Gli smart contract di Algorand consentono di realizzare applicazioni di pagamento totalmente personalizzate. Un esempio di questa tipologia di applicazioni è l'Algorand Mobile Wallet che consente agli

utenti di ricevere e inviare Algo. Inoltre è possibile fare lo stesso anche con delle stablecoin sotto forma di Algorand Standard Asset.

- **Mercati di previsione**: Gli smart contract possono avere un ruolo fondamentale anche nel settore dei mercati di previsione i quali consentono di acquistare o vendere azioni a seguito di un evento. In questo tipo di mercati gli utenti piazzano "scommesse" sul fatto che un determinato evento si verifichi. Utilizzando gli smart contract è possibile implementare soluzioni in cui il contratto assume il compito, una volta che l'evento si è verificato, di controllare e inviare fondi a tutti coloro che avevano scommesso su quell'evento.

- **Mercati online**: I mercati online odierni sono basati su autorità centralizzate che acquisiscono commissioni sulle transazioni che vengono effettuate. Attraverso l'uso di smart contract è possibile creare un mercato basato su utenti allo stesso livello con l'opportunità di vendere i propri prodotti senza la necessità di cedere costi di commissione a terze parti. Spesso inoltre i mercati centralizzati peccano da un punto di vista di sicurezza, mentre gli smart contract garantiscono agli utenti una maggiore affidabilità [2].

3.2 Smart Contract Stateless

Nel capitolo precedente, più precisamente all'interno della sezione relativa alle transazioni, sono state analizzate le diverse tipologie di firma, necessarie per l'autorizzazione dei trasferimenti in Algorand. In Algorand tutte le transazioni che vengono inviate devono passare attraverso la fase di autorizzazione e devono essere necessariamente firmate, o da un singolo account o da un account multifirma. Lo scopo principale degli smart contract senza stato è quello di rimpiazzare l'autorità di firma. Questo viene fatto attraverso la così detta Firma Logica (LogicSignature). L'utilizzo della LogicSignature fa si che la transazione firmata con questa modalità passi attraverso i controlli dettati dalla logica del contratto e, al termine di questi, venga emesso un esito che determina l'accettazione o il rifiuto della transazione. La LogicSig per poter essere utilizzata in una transazione deve prima essere considerata valida, e ciò avviene se si verifica uno dei seguenti casi:

- La Sig contiene una firma valida del programma dal mittente della transazione.

- La Sig contiene una multi firma valida del programma dall'account a firma multipla che sta inviando la transazione.

- L'hash del contratto coincide con l'indirizzo del mittente.

Le firme logiche, indicate come LogicSig sono strutturate come descritto nella Fig. 3.1.

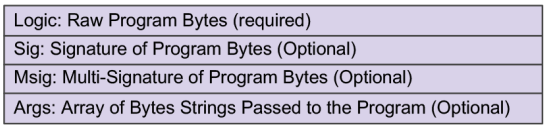

Figura 3.1. Struttura della firma logica

Source: https://developer.algorand.org/docs/features/asc1/stateless/modes/

La scrittura della logica dei contratti viene realizzata attraverso l'uso del linguaggio TEAL, il quale approfondiremo nei paragrafi successivi. Gli Smart Contract Stateless hanno due modalità d'uso che sono state analizzate qui di seguito, account di contratto e firma delegata [4].

3.2.1 Contract Account

I Contract Account sono la tipologia di contratto più utilizzata e agiscono in modo simile ai conti di deposito a garanzia (Escrow). Per utilizzare questo tipo di contratto stateless il programma deve essere compilato, ciò avviene attraverso l'uso degli SDK, come Python, Go o Javascript, oppure utilizzando GOAL, ovvero l'interfaccia a linea di comando che consente di interagire con l'istanza del software di Algorand. A seguito della compilazione viene generato un indirizzo univoco che identificherà il contratto. L'indirizzo che gli viene assegnato garantisce al contratto tutte le funzionalità di un normale account, ciò gli consente di avere un suo specifico saldo sulla blockchain e di conseguenza di ricevere e inviare Algo o Algorand Standard Asset. I passaggi logici relativi alla creazione di un contract account sono illustrati nella Fig. 3.2.
L'unica differenza con un normale account Algorand risiede nel modo in cui le transazioni in uscita vengono approvate. L'approvazione di transazioni di spesa da un Contract Account vengono determinate dalla logica scritta all'interno del contratto stesso. Affinché si possa inviare una transazione da un account di contratto è necessario generare una transazione che rispetta le condizioni imposte dalla logica TEAL, e infine utilizzare il codice TEAL compilato come firma logica.

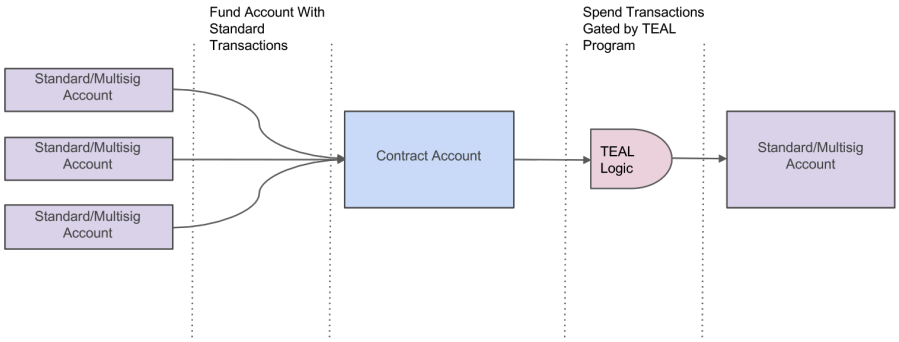

Figura 3.2. TEAL Contract Account

Source: https://developer.algorand.org/docs/features/asc1/stateless/modes/

Di seguito è descritto il processo d'uso per la creazione e l'invio di una transazione da parte di un Contract Account:

- Caricare i byte del programma nell'SDK.
- Creare la LogicSig derivante dal programma.
- Generare la transazione.
- Impostare l'indirizzo del contratto nel campo `from` della transazione.
- Firmare la transazione con la LogicSig.
- Inviare la transazione alla rete.

Nella Fig. 3.3 che segue viene descritto lo stesso processo a livello logico.

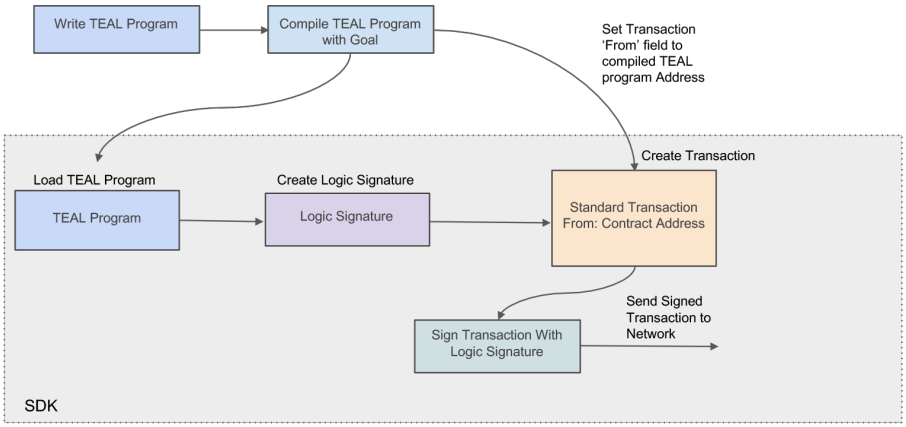

Figura 3.3. Transazione da un Contract Account

Source: https://developer.algorand.org/docs/features/asc1/stateless/sdks/

3.2.2 Delegated Signature

Gli smart contract di tipo stateless possono essere utilizzati per la delegazione dell'autorità di firma. A differenza dei contract account in questo caso la logica del contratto viene firmata da un singolo account o da un account multifirma attraverso la chiave privata. In seguito alla validazione della firma questa può essere distribuita ed utilizzata da altri account al fine di inviare transazioni, basate sulla logica del contratto, per effettuare prelievi di Algo o di ASA. Lo schema logico relativo alla creazione di un contratto di tipo Delegated Signature è illustrato nella Fig. 3.4. Una situazione esemplificativa di utilizzo di un contratto a firma delegata potrebbe essere il caso in cui un datore di lavoro crea un contratto delineando le condizioni relative agli stipendi dei suoi impiegati. Dopo aver definito tutte le condizioni necessarie attraverso il linguaggio TEAL, il datore valida la LogicSig e poi la rende disponibile ai suoi dipendenti affinchè possano utilizzarla periodicamente per ritirare i propri stipendi in modo autonomo a patto di rispettare le condizioni imposte dalla logica.

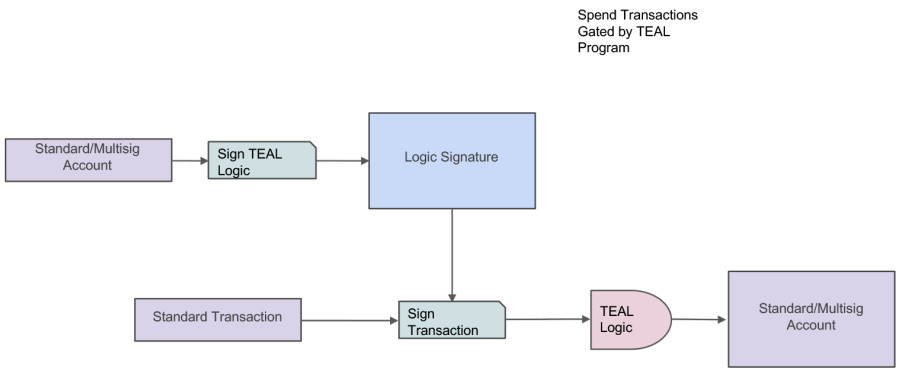

Figura 3.4. TEAL Delegated Signature

Source: https://developer.algorand.org/docs/features/asc1/stateless/modes/

Di seguito è descritto il processo d'uso per la creazione e l'invio di una transazione con firma delegata:

- Caricare i byte del programma nell'SDK.
- Creare la LogicSig derivante dal programma.
- Firmare la LogicSig con la chiave privata di un account specifico.
- Generare la transazione.
- Impostare il campo `from` della transazione con l'indirizzo dell'account che ha firmato la logica.

- Firmare la transazione con la LogicSig.
- Inviare la transazione alla rete [4].

Nella Fig. 3.5 che segue viene descritto lo stesso processo a livello logico.

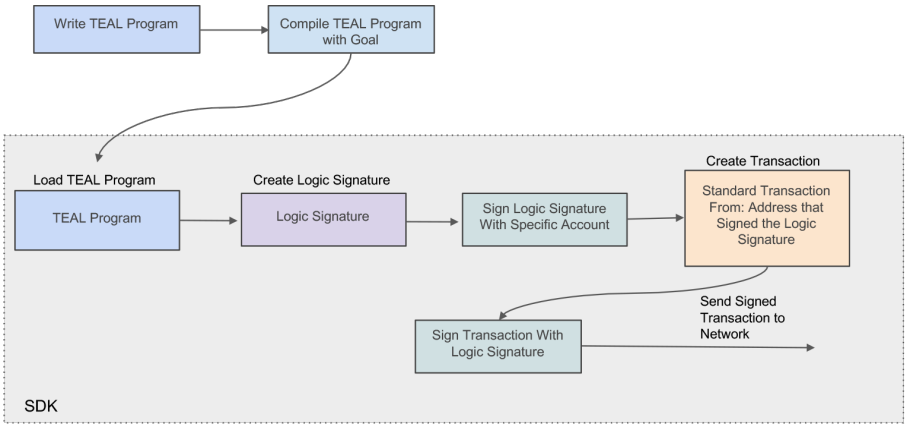

Figura 3.5. Transazione con firma delegata

Source: https://developer.algorand.org/docs/features/asc1/stateless/modes/

3.3 Smart Contract Stateful

La seconda tipologia di contratti che Algorand offre sono gli Stateful Smart Contract, ovvero, smart contract con stato. La caratteristica principale di questi contratti è la capacità di mantenere informazioni all'interno di uno stato memorizzato on-chain. Lo stato inoltre si differenzia in locale e globale; lo stato locale memorizza informazioni specifiche relative ad un singolo account; lo stato globale mantiene i dati relativi a tutti gli account che partecipano al contratto. Un tipico esempio di utilizzo può essere quello relativo ad un'applicazione per una raccolta fondi. Al momento della creazione viene predisposto lo stato globale che memorizzerà il totale delle donazioni effettuate, ed uno stato locale per mantenere i versamenti effettuati da ogni singolo utente. Ogni qualvolta un account effettua una donazione, viene modificato lo stato locale relativo all'utente aggiornando il valore della sua donazione e lo stato globale a cui verrà sommato il saldo del nuovo versamento.
I contratti stateful rappresentano la colonna portante delle applicazioni in esecuzione sulla blockchain di Algorand e consentono di realizzare moltissimi casi d'uso differenti. Inoltre è possibile creare applicazioni complesse grazie alla possibilità di combinare contratti Stateful con contratti Stateless attraverso l'uso degli Atomic Transfer [4].
I contratti stateful sono composti da due programmi scritti in linguaggio TEAL, ognuno dei quali svolge un ruolo distinto:

- **ApprovalProgram**: Programma TEAL che implementa la logica del contratto ed ha il compito di gestire tutte le Application Call, ovvero, le richieste utilizzate per interagire con contratti Stateful, a parte le Clear Call che sono descritte nel prossimo punto. Le chiamate che questo programma riceve, proprio come succede con i contratti stateless, vengono approvate o rifiutate in base alla logica TEAL.

- **ClearStateProgram**: Un account prima di poter utilizzare un contratto stateful necessita di inviare una richiesta di Opt-in all'applicazione. In seguito all'invio di questa richiesta viene aggiunto al record di bilancio dell'account il valore relativo allo stato locale. Lo scopo del ClearStateProgram è quello di eliminare la partecipazione di un account da un contratto stateful quando questo viene richiesto. L'eliminazione consiste semplicemente nella cancellazione dello stato locale dal record di bilancio dell'account che ha effettuato la Clear Call, ovvero, la chiamata al ClearStateProgram. Anche il suo funzionamento si basa sulla logica TEAL.

L'architettura degli Algorand Smart Contract Stateful è descritta nel dettaglio nella Fig. 3.6.
La creazione di due programmi differenti consente di separare la logica del contratto dalle operazioni di cancellazione, così da gestire le diverse situazioni in modo indipendente.
Le chiamate che vengono utilizzate per l'interazione con contratti stateful sono un particolare tipo transazione chiamate ApplicationCall. Queste transazioni vengono utilizzate per svolgere operazioni differenti all'interno di un contratto e sono esposte qui di seguito:

- **NoOp**: Chiamata generica per eseguire l'ApprovalProgram.

- **OptIn**: Chiamata per aderire al contratto.

- **DeleteApplication**: Chiamata per eliminare l'applicazione.

- **UpdateApplication**: Chiamata che consente di modificare il codice TEAL mantenendo lo stesso ID dell'applicazione.

- **CloseOut**: Chiamata utilizzata da un account per eliminare la partecipazione ad un'applicazione, cancellando lo stato locale dal record di bilancio. Questa transazione si basa sulla logica TEAL, quindi per andare a buon fine deve rispettare le condizioni imposte dalla logica del contratto.

- **ClearState**: Chiamata simile alla precedente, con la differenza che questa viene eseguita sempre poiché non necessità di approvazione da parte del

programma. Questa è l'unica tipologia di transazione che viene gestita dal ClearStateProgram [4].

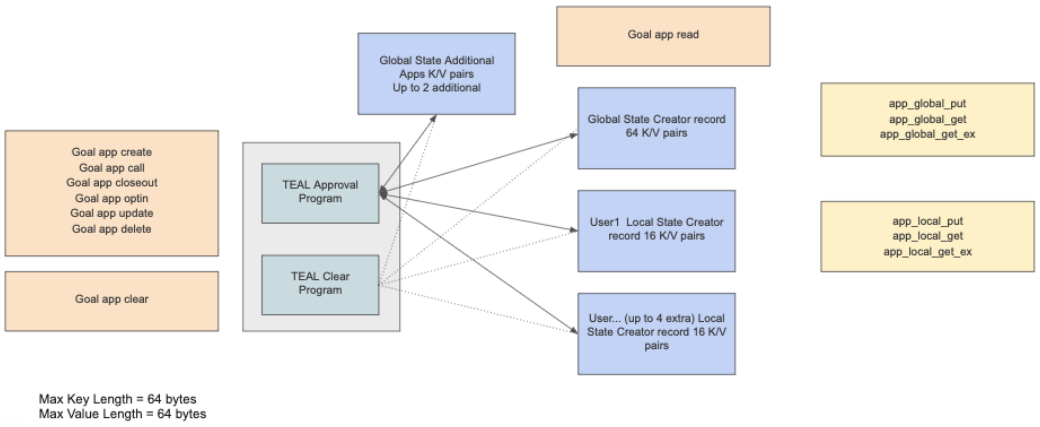

Figura 3.6. Architettura Smart Contract Stateful

Source: https://developer.algorand.org/docs/features/asc1/stateful/

3.4 TEAL (Transaction Execution Approval Language)

Gli smart contract di Algorand sono implementati con un particolare linguaggio chiamato TEAL, acronimo di Transaction Execution Approval Language. TEAL è un linguaggio di tipo assembly basato su stack e non-Turing-completo, esso di conseguenza non supporta ne iterazioni ne ricorsioni ma consente solo di eseguire dei forward branch. TEAL supporta due tipi di dato, interi a 64 bit o stringhe di byte. E' un linguaggio basato su codici operativi (OpCodes) ed elabora una singola linea di programma alla volta eseguendo operazioni di push e pop sullo stack. TEAL non è in grado di generare o modificare transazioni. L'unica operazione che gli è concessa compiere è quella di analizzare le transazioni che gli vengono sottoposte, al fine di approvare o rifiutare queste ultime se vengono rispettate le condizioni da esso descritte. Tra le altre cose che TEAL non è in grado fare c'è la previsione del momento in cui la transazione viene inviata, e la visione dei bilanci in termini di Algo e di Asset.

Questo linguaggio permette l'uso di diversi operatori che agiscono sui dati nello stack, consente il passaggio di argomenti dalla transazione al programma, possiede uno scratch space dove poter memorizzare valori temporanei, e permette l'accesso ai campi di transazioni singole o raggruppate. L'architettura del linguaggio TEAL è dettagliatamente descritta nella Fig. 3.7.

Gli Opcode a cui TEAL può accedere si differenziano in base al tipo di contratto

che si vuole creare. La tipologia di contratto viene definita dall'attributo `Mode`, il quale se impostato a `Signature` indica l'implementazione di un contratto stateless, mentre se impostato ad `Application` indica la creazione di un contratto di tipo stateful.

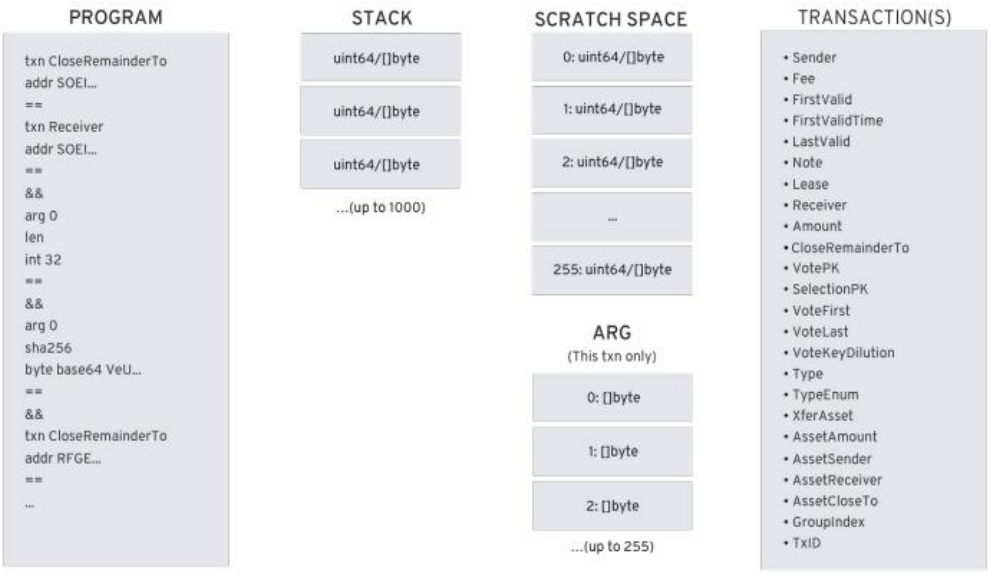

Figura 3.7. Architettura TEAL

Source: https://developer.algorand.org/docs/features/asc1/teal/

La logica di funzionamento di questo linguaggio è molto semplice. Vengono immessi valori all'interno dello stack a due a due attraverso operazioni di push. Successivamente viene applicato un operatore di confronto ai due valori inseriti e da questa operazione ne risulta un valore booleano, true o false, che verrà a sua volta salvato sullo stack. Se tutti i controlli previsti nel programma restituiscono il valore true allora questo sarà anche l'ultimo valore presente nello stack e in quel caso la transazione verrà approvata e inviata. Di seguito sono esposte le componenti principali del linguaggio TEAL e alcune caratteristiche del suo funzionamento logico:

- **Recupero delle proprietà della transazione**: Per accedere alle proprietà delle transazioni inviate singolarmente viene utilizzato il comando `txn`, mentre per le transazioni relative ad un trasferimento atomico viene usato il comando `gtxn` seguito da un valore relativo all'indice della transazione a cui si vuole far riferimento all'interno del gruppo. Ad esempio, per accedere alla proprietà

`Amount` della seconda transazione di un gruppo, il comando necessario è il seguente: `gtxn 1 Amount`.

- **Pseudo Opcode**: TEAL fornisce anche la possibilità di utilizzare degli pseudo opcode che semplificano la scrittura del codice. Un esempio di questi opcode sono i comandi `int`, `byte`, e `addr`, i quali semplificano la scrittura rispettivamente di valori interi, stringhe e indirizzi.

- **Operatori**: TEAL consente l'utilizzo di numerosi operatori di confronto i quali, come già detto in precedenza, restituiscono un valore di true o false che viene inserito in cima allo stack. Alcun esempi di operatori sono `<`, `>`, `==`, `!=`, `<=`, `>=`, `&&`.

- **Passaggio di argomenti**: Il passaggio di parametri a programi TEAL avviene attraverso l'uso degli SDK o di Goal. All'interno del programma si accede a questi argomenti attraverso il comando `arg` seguito dal numero relativo all'indice dell'argomento. Ad esempio il comando `arg 0` fa riferimento al primo argomento nella lista dei parametri. Inoltre tutti gli argomenti in TEAL devono essere array di byte.

- **Utilizzo Scratch Space**: Lo scopo dello scratch space è quello di memorizzare valori che potranno essere utilizzati successivamente. L'utilizzo di questa area si costituisce di due funzioni, `load` e `store`, ovvero, lettura e scrittura. Il comando `store 1` effettua il pop dell'elemento con indice 1 nello stack e lo inserisce nello slot 1 dello scratch space. Il comando `load 1` recupera il valore dello slot 1 dallo scratch space e lo reinserisce nello stack [4].

Capitolo 4

Progettazione e Realizzazione

Al fine di studiare nel modo più approfondito possibile la piattaforma Algorand, e nello specifico i suoi smart contract, si è scelto di applicare la tecnologia analizzata ad un caso di studio, esposto ed analizzato in questo capitolo. Lo scenario ci consentirà, attraverso differenti casi d'uso, di comprendere a pieno gli Algorand Smart Contract e le loro modalità di utilizzo.

4.1 Progettazione del caso di studio

Lo scenario scelto, come già accennato nel capitolo introduttivo, rientra nell'ambito dell'economia circolare. Questo modello economico pone come suo obiettivo principale la riduzione dell'uso di nuove risorse, privilegiando una politica di riciclo e riutilizzo.
Nelle grandi città, come ad esempio Roma, sorge spesso un problema legato alla presenza di rifiuti ingombranti nelle strade, o anche più semplicemente alla presenza di rifiuti non adatti alla raccolta da parte degli operatori ecologici. Questo problema, oltre ad essere legato ad un fattore puramente estetico, provoca numerose difficoltà da un punto di vista logistico, creando ostacoli per pedoni e veicoli. Il progetto realizzato mira a sensibilizzare l'utente riguardo questo argomento spronandolo al corretto smaltimento di queste tipologie di rifiuti presso i numerosi punti di raccolta presenti nelle aree urbane.
Per far si che le persone siano stimolate nel compiere le giuste azioni si è pensato di garantire una sorta di premio in seguito al corretto riciclo di rifiuti ingombranti. Le ricompense derivate dal corretto smaltimento di rifiuti consistono nell'attribuzione di monete virtuali rappresentate da Algorand Standard Asset. Questi asset potranno in seguito essere convertiti e utilizzati per l'acquisto di servizi pubblici di base, come ad esempio biglietti per la metropolitana o per l'autobus.
Questo progetto prevede che l'utente possieda un Algorand wallet in cui verranno

mantenute le disponibilità dei vari asset coinvolti nel caso di studio, ognuno con il suo preciso scopo e campo di utilizzo. Gli asset utilizzati all'interno dello scenario sono i seguenti:

- **EcoAsset**: Asset distribuito come ricompensa in seguito al riciclo di rifiuti ingombranti. La denominazione per una sua unità è ECO.
- **MetroAsset**: Asset ottenibile mediante lo scambio di ECO. Viene utilizzato per l'acquisto di titoli di viaggio. La denominazione di una sua unità è METRO.
- **TransportAsset**: Asset ottenibile mediante lo scambio di ECO. Viene utilizzato per richiedere il ritiro a domicilio di rifiuti ingombranti. La denominazione di una sua unità è TRAN.

Ognuna di queste risorse verrà esposta dettagliatamente in seguito, descrivendone le configurazioni e i relativi tassi di cambio.
Nello scenario che stiamo descrivendo prendono parte diversi attori, quali l'utente, l'operatore ecologico, e l'addetto ai trasporti. Questi potranno svolgere determinate azioni, spesso interagendo anche tra di loro. Tutti i casi d'uso sono stati realizzati con la finalità di approfondire lo studio delle funzionalità offerte dagli Algorand Smart Contract. Nella tabella seguente è illustrata l'associazione tra i casi d'uso utilizzati e gli aspetti relativi agli ASC che espongono:

Tabella 4.1. Tabella riassuntiva delle funzionalità realizzate mediante i casi d'uso

Id	**Titolo**	**Funzionalità**
1	Ricicla Ingombranti	Interazione contratti Stateful e Stateless
2	Acquista METRO	Combinazione tra Atomic Transfer e smart contract
3	Acquista ECO	Interoperabilità tra Algo, Asset e smart contract
4	Acquista TRAN	Interrogazione di contratti Stateful
5	Ritiro a Domicilio	Creazione scenario escrow tramite contratti

Gli scenari d'interesse sono descritti brevemente qui di seguito e sono inoltre illustrati mediante un diagramma UML nella Fig. 4.1.

1. Riciclare rifiuti ingombranti in cambio di EcoAsset.
2. Acquistare MetroAsset scambiandoli con EcoAsset.
3. Acquistare EcoAsset scambiandoli con Algo.
4. Acquistare TransportAsset scambiandoli con EcoAsset.
5. Ritiro di rifiuti ingombranti a domicilio.

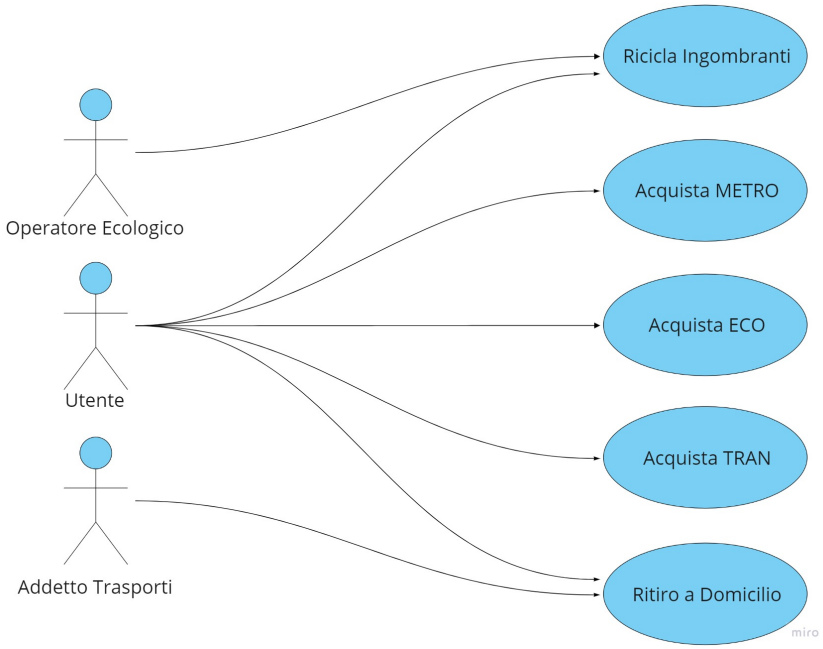

Figura 4.1. Casi d'Uso

Entriamo ora nel dettaglio degli use case appena citati, analizzando dettagliatamente ciascuno di essi nelle sottosezioni che seguono.

4.1.1 Caso d'uso 1: Riciclare rifiuti ingombranti

Questo primo caso d'uso è stato realizzato al fine di mostrare le possibilità di interazione tra contratti stateless e stateful e l'utilizzo combinato di contract account e Algorand Standard Asset. lo scenario coinvolge due attori, l'utente che consegna i rifiuti ingombranti e l'operatore ecologico che si occupa del loro smaltimento. In primo luogo l'utente si reca presso una delle tante isole ecologiche a disposizione portando con se i rifiuti ingombranti che desidera depositare. A questo punto l'operatore ecologico effettua delle misurazioni su ciò che l'utente ha consegnato. Tali misurazioni vengono eseguite sui singoli rifiuti consegnati all'operatore. Prima di tutto il rifiuto ingombrante viene pesato e in base al suo peso viene assegnato uno specifico valore, denotato dai seguenti criteri:

- Se il peso è compreso tra 0 e 10 chilogrammi il valore assegnato è 1.
- Se il peso è compreso tra 11 e 50 chilogrammi il valore assegnato è 2.
- Se il peso è compreso tra 51 e 100 chilogrammi il valore assegnato è 3.
- Se il peso è compreso tra 101 e 200 chilogrammi il valore assegnato è 4.

- Se il peso è superiore a 200 chilogrammi il valore assegnato è 5.

Dopo la fase di pesatura l'operatore attribuisce un valore di riciclabilità in un range che va da 1 a 5, il quale indicherà quanto i materiali consegnati dall'utente sono predisposti al riutilizzo. In base ai valori derivanti da queste operazioni viene calcolato il compenso, consistente in EcoAsset (ECO), che verrà in seguito consegnato all'utente. Il premio viene ottenuto semplicemente moltiplicando il valore relativo alla pesatura e il valore di riciclabilità. Il maggior compenso che è possibile ricevere ad ogni singola consegna è di 25 ECO.
In seguito alla fase di valutazione si passa a quella in cui avviene il trasferimento del compenso all'utente e a cui prendono parte gli Algorand Smart Contract. Per la progettazione di questo caso d'uso sono stati utilizzati due smart contract, il primo è un contratto di tipo stateless, più precisamente un contract account, mentre il secondo è un contratto di tipo stateful. In seguito alla compilazione del contratto stateless viene generato un indirizzo che, mediante una transazione di tipo Asset Configuration (`acfg`), viene assegnato all'attributo `Clawback address` dell'EcoAsset. L'operazione appena descritta consente al contract account di poter revocare unità dell'EcoAsset e di inviarle ad un altro indirizzo, firmando lui stesso la transazione. Ciò significa che non è il creatore dell'EcoAsset ad inviare il compenso all'utente, bensì lo smart contract attraverso una transazione di revoca. Questo processo necessita anche che l'indirizzo relativo all'account creator dell'EcoAsset sia oggetto di una transazione di tipo freeze (`afrz`), la quale fa si che l'account venga privato della possibilità di inviare o ricevere unità di ECO. Le operazioni appena descritte hanno lo scopo di controllare la diffusione di ECO, regolamentandone la distribuzione mediante il consenso della logica TEAL derivata dal contratto stateless.
L'invio del premio all'utente avviene con una transazione appartenente ad un gruppo atomico che viene sottoposto alla logica TEAL del contract account. Al trasferimento atomico appena citato appartiene anche una seconda transazione, o più precisamente un Application Call, indirizzata ad un contratto di tipo stateful. Quest'ultimo mantiene alcune informazioni divise nello stato globale e locale:

- **Stato Globale**: Mantiene l'indirizzo relativo al Creator del contratto, un contatore relativo al totale di consegne effettuate da tutti gli utenti, e un contatore indicante quanti rifiuti ingombranti sono stati depositati per ogni livello di riciclabilità.
- **Stato Locale**: È relativo ad un singolo utente e mantiene un conteggio totale delle consegne da lui effettuate.

La transazione di tipo Application Call viene firmata dall'utente e contiene tra i suoi argomenti una stringa e il valore relativo al livello di riciclabilità. Questa chiamata,

indirizzata al contratto stateful, fa si che venga incrementato il totale di consegne effettuate dagli utenti, il contatore del livello di riciclo specificato come argomento della transazione, e il conteggio di consegne dell'utente.
All'interno del contratto stateless vengono effettuati controlli anche sull'Application Call indirizzata al contratto stateful il che rappresenta proprio l'interazione tra queste diverse tipologie di contratto. Per concludere, se il trasferimento atomico supera la fase di approvazione imposta dalla logica di contratto, l'utente riceve all'interno del suo wallet il compenso in ECO.
Nella Fig. 4.2 viene illustrato il diagramma di comunicazione relativo a questo caso d'uso. In principio `TransactionsManager` riceve il risultato della compilazione di `RecycleStatelessContract`(`program`) e l'ApplicationId(`app_id`) relativo a `RecycleStatefulContract`. Proseguendo l'`Operatore Ecologico` invoca `TransactionsManager` mediante `execute_txn()`. Quest'ultimo richiede la firma logica a `TransactionClass` fornendo `program` come parametro. In seguito genera e firma la prima transazione(`txn_0`) mediante la firma logica(`lsig`) ottenuta. Proseguendo `TransactionsManager` genera la seconda transazione(`txn_1`) e richiede l'autorizzazione dell'utente, il quale la firma mediante il metodo `sign(private_key)` di `TransactionClass`. In conclusione `TransactionsManager` invoca `Node`, il quele invia le transazioni alla blockchain(`AlgorandBlockchain`) mediante un trasferimento atomico.

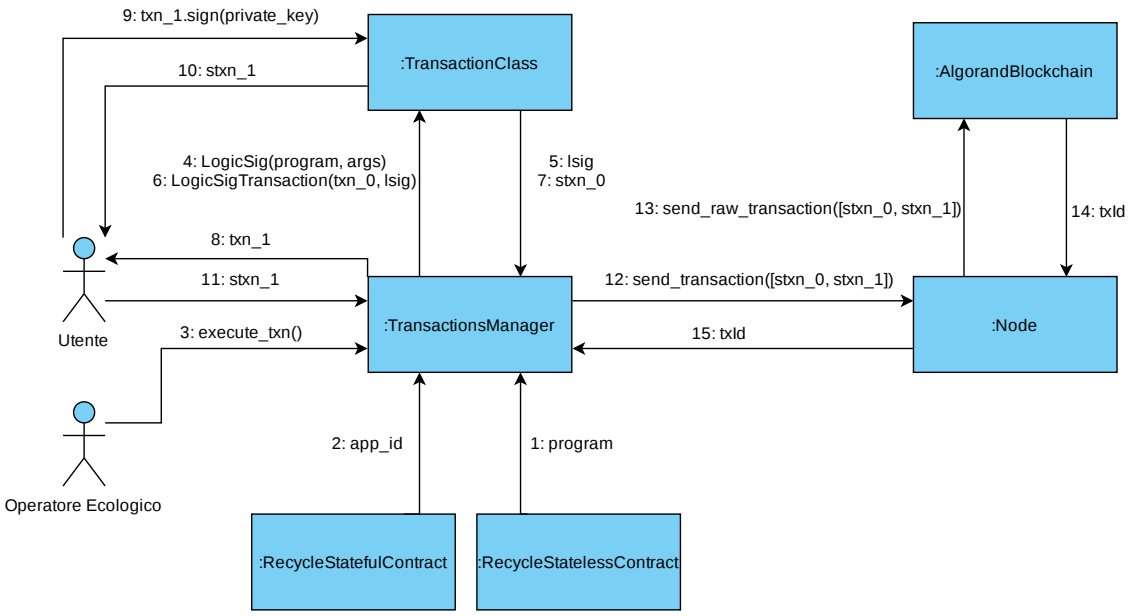

Figura 4.2. Diagramma di comunicazione Use Case 1

4.1.2 Caso d'uso 2: Acquistare MetroAsset

Questo caso d'uso è stato progettato al fine di studiare le potenzialità degli Atomic Transfer di Algorand combinati con un contratto stateless. L'obiettivo in questo scenario è quello di scambiare unità di EcoAsset con unità di MetroAsset. I MetroAsset possono essere utilizzati al fine di acquistare titoli di viaggio per il trasporto urbano. Il tasso di cambio tra i due asset coinvolti è di 1 a 15, ovvero, 15 ECO = 1 METRO. Un METRO consente l'acquisto di un singolo titolo di viaggio. Nello scenario che stiamo descrivendo viene coinvolto un solo contratto di tipo stateless compilato come un contract account, il quale viene fornito di una certa quantità di METRO al momento della sua creazione. Lo scambio delle risorse necessita semplicemente dell'utilizzo dell'Algorand wallet dell'utente. Quest'ultimo infatti non deve far altro che richiedere lo scambio e, se il suo saldo di EcoAsset è sufficiente, riceverà il relativo compenso in MetroAsset. L'operazione appena descritta utilizza un trasferimento atomico che coinvolge due transazioni. La prima transazione prevede il trasferimento di 1 METRO dal contract account verso l'indirizzo del richiedente, mentre la seconda prevede l'invio di 15 ECO da parte dell'utente verso uno specifico indirizzo relativo al fornitore del servizio. La prima transazione del gruppo atomico viene firmata dal contract account mediante una firma logica, questo implica che entrambe le transazioni passano attraverso una fase di approvazione dettata dal contratto. Se il trasferimento rispetta le condizioni imposte lo scambio avviene in modo simultaneo, altrimenti nessuna transazione viene inviata.
Qualora il trasferimento atomico venga approvato gli EcoAsset ricevuti dal destinatario della seconda transazione vengono resi nuovamente disponibili per lo scenario descritto al punto 4.1.1.
Nella Fig. 4.3 è illustrato il diagramma di comunicazione rappresentante l'interazione tra i moduli coinvolti in questo caso d'uso. Inizialmente `TransactionsManager` riceve `program` come risultato della compilazione di `AssetExchangeContract`. Proseguendo, l'`Utente` invoca `TransactionsManager` mediante `groups_transactions()`. Questo richiede la firma logica a `TransactionClass` fornendo `program` come parametro. In seguito genera e firma la prima transazione, ovvero, `txn_0`, mediante la firma logica(`lsig`) ottenuta. Dopodichè genera la seconda transazione(`txn_1`) e richiede all'utente di autorizzarla. Quest'ultimo firma la transazione invocando `TransactionClass` mediante la chiamata `txn_1.sign(private_key)`, nella quale specifica la propria chiave privata come parametro. Attraverso tale processo, l'utente ottiene la transazione autorizzata (`stxn_1`). In conclusione `TransactionsManager` invoca `Node`, mediante trasferimento atomico, invia le transazioni alla blockchain(`AlgorandBlockchain`).

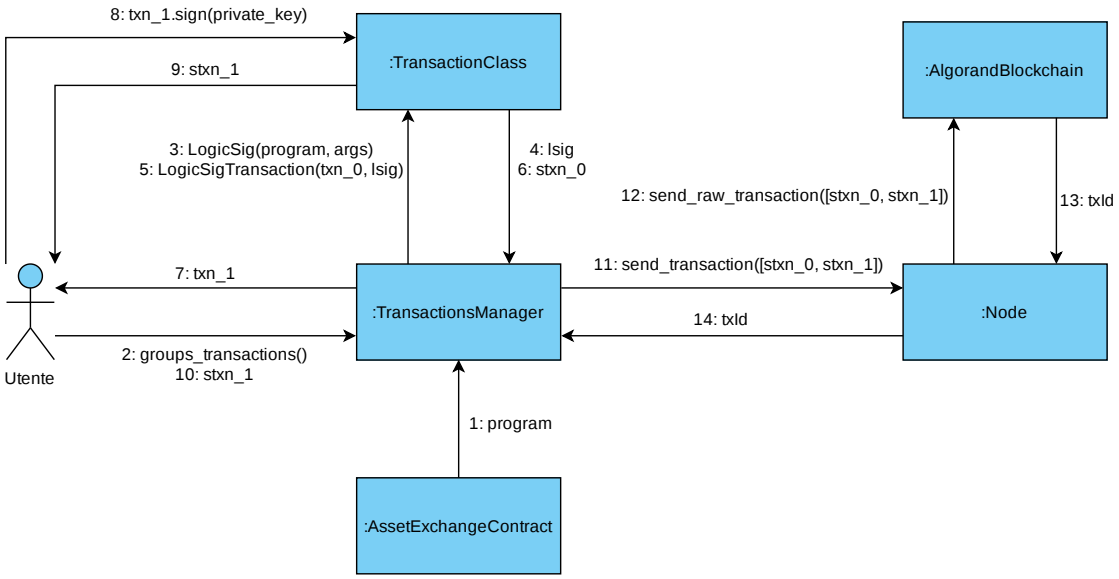

Figura 4.3. Diagramma di comunicazione Use Case 2

4.1.3 Caso d'uso 3: Acquistare EcoAsset

Lo scenario illustrato qui di seguito consente di mostrare la facilità con cui è possibile combinare l'utilizzo di Algo nativi e asset attraverso l'uso degli Algorand smart contract.
Lo scenario coinvolge esclusivamente l'utente e per illustrarlo prendiamo in considerazione un esempio pratico. Supponiamo che un utente voglia acquistare un titolo di viaggio equivalente in 1 METRO e che il suo saldo in ECO ammonti a 10. Se provasse ad acquistare MetroAsset mediante l'operazione descritta nella sottosezione 4.1.2, la sua richiesta verrebbe rifiutata a causa della mancanza di fondi. In tale situazione viene offerta la possibilità all'utente di acquistare unità di EcoAsset pagando direttamente in Algo. Questa operazione rende l'EcoAsset una moneta fungibile. La fungibilità è la caratteristica di un bene di poter essere scambiato con un altro dal medesimo valore [9]. Dato il valore economico a cui l'Algo è legato (quotazione che oscilla intorno a 1.20€ [12]), lo scambio conferirà anche all'EcoAsset un valore esprimibile nei medesimi termini. Si è scelto di conferire un tasso di cambio che fosse consono al valore della moneta Algo e al prezzo di acquisto di un titolo di viaggio, 1 Algo = 10 ECO.
Nel momento in cui l'utente desidera acquistare EcoAsset viene richiesto di inserire l'ammontare di unità che desidera ricevere. Da queste viene calcolato il prezzo in Algo richiesto all'utente. Viene generato un trasferimento atomico che coinvolge due transazioni dedite allo scambio, la prima dal contract account verso l'utente con l'ammontare di ECO richiesto, la seconda dall'utente verso il fornitore di servizi con

il totale di Algo precedentemente calcolato. La prima transazione viene autorizzata mediante la firma logica generata dal contratto stateless sottoponendo di conseguenza la transazione ai controlli dettati dal codice TEAL. In caso di superamento di tali condizioni le transazioni vengono inviate in modo sincrono.
La Fig. 4.4 illustra i moduli coinvolti nelle operazioni. Il ciclo di esecuzione mostrato nel diagramma è analogo a quello esposto nel caso d'uso precedente(4.1.2). Le uniche differenze si riscontrano nell'oggetto rappresentante il contratto coinvolto, che in questo caso prende il nome di `ExchangeAlgoContract`, e nel metodo utilizzato per l'invocazione di `TransactionsManager`, ovvero, `exchange()`.

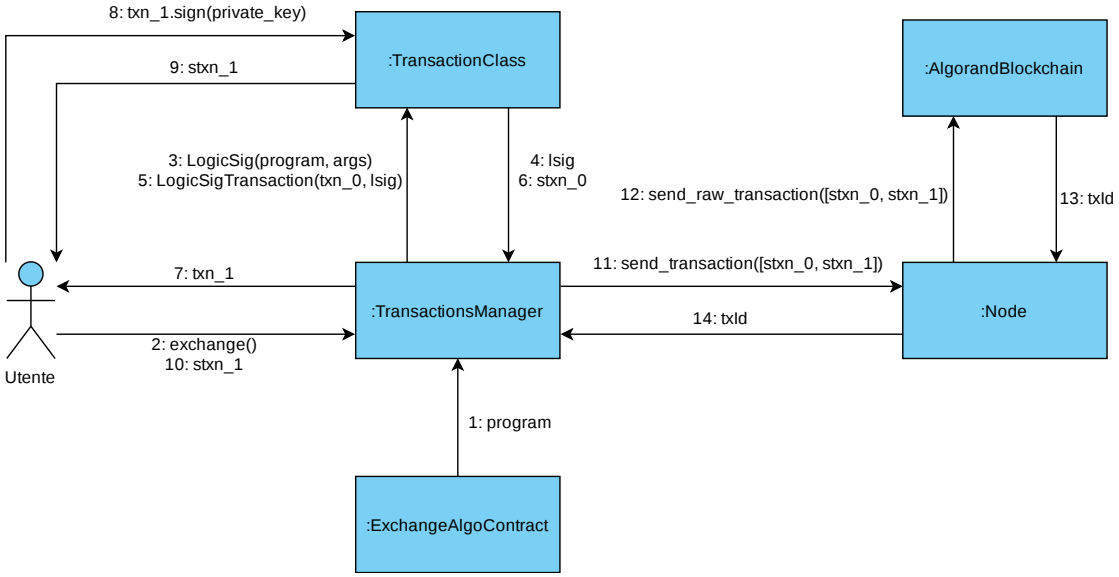

Figura 4.4. Diagramma di comunicazione Use Case 3

4.1.4 Caso d'uso 4: Acquistare TransportAsset

Lo scenario descritto di seguito risulta essere molto simile al caso d'uso già esposto nella sottosezione 4.1.2, di conseguenza eviteremo ripetizioni non necessarie. Nonostante la premessa, questo caso d'uso risulta molto importante al fine di studiare la possibilità di interazione con contratti di tipo stateful e più in generale di visionare quelle che sono le funzionalità che questi possono implementare.
Anche questo caso d'uso coinvolge il singolo utente e consente a quest'ultimo di scambiare EcoAsset con TransportAsset, ad un tasso denotato dalla seguente equazione, 300 ECO = 1 TRAN. Ciò che differenzia questo scenario da quello che illustra la possibilità di scambio tra ECO e METRO, è l'interazione con il contratto stateful già presente nel caso d'uso 4.1.1. In questo contesto sono coinvolte tre diverse transazioni, due per garantire lo scambio di ECO e TRAN ed una per l'interrogazione del contratto stateful. Al fine di porre limitazioni sulle possibilità di utilizzo del caso

d'uso che stiamo descrivendo, è stato richiesto all'utente una verifica del conteggio relativo alle consegne di rifiuti ingombranti effettuate. Ciò che viene richiesto è che quest'ultimo abbia effettuato un totale di almeno 10 depositi presso le diverse isole ecologiche. La terza transazione del gruppo atomico ha proprio lo scopo di richiedere questa verifica, infatti questa transazione consiste in un'Application Call diretta al contratto stateful. Quest'ultimo a seguito della chiamata ricevuta restituirà `true` se la condizione viene rispettata, `false` altrimenti. In aggiunta al contratto stateful, anche in questo scenario prende parte un contract account incaricato di approvare o rifiutare il trasferimento atomico in base alla logica TEAL che implementa.
Anche in questo caso è stato illustrato un diagramma di comunicazione che descrive l'interazione tra i moduli coinvolti nello scenario(Fig 4.5). `TransactionsManager` ottiene `program` e `app_id`, rispettivamente il risultato della compilazione di `TransportContract` e l'`ApplicationId` di `RecycleStatefulContract`. In seguito `TransactionsManager` genera le transazioni costituenti il gruppo atomico e richiedendone l'autorizzazione a `TransactionClass`. La prima(`txn_0`) richiede la firma logica(`lsig`) mentre la seconda(`txn_1`) e la terza(`txn_2`) l'autorizzazione dell'utente. Infine `TransactonsManager` comunica con `Node`, il quale invia le transazioni ad `AlgorandBlockchain`.

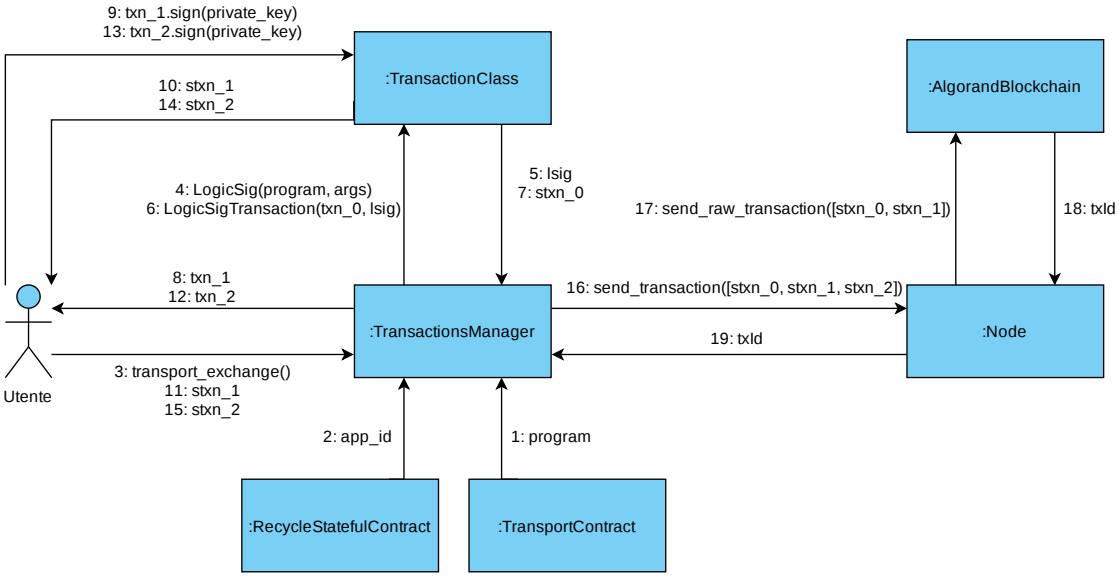

Figura 4.5. Diagramma di comunicazione Use Case 4

4.1.5 Caso d'uso 5: Richiedere ritiro ingombranti a domicilio

Per favorire lo studio degli Algorand Smart Contract si è scelto di implementare un caso d'uso classico nell'economia decentralizzata, ovvero, il deposito a garanzia, o escrow. Descritto già precedentemente nel capitolo 3, l'escrow mira ha garantire fiducia ed equità tra cliente e fornitore del servizio. Al fine di rappresentare un ac-

cordo gestito tramite deposito a garanzia si è scelto di utilizzare lo scenario seguente. Nel caso in cui un utente abbia necessità di smaltire un rifiuto ingombrante particolarmente inadatto al trasporto, può usufruire di un servizio di ritiro domiciliare gestito da addetti di competenza. Questo scenario coinvolge due attori, l'utente richiedente il servizio, e un operatore che si occupa del trasporto. Il caso d'uso richiede un pagamento tramite TRAN, e coinvolge un solo contratto stateless di tipo contract account avente il ruolo di deposito a garanzia.
Inizialmente l'utente fa richiesta per il ritiro alla ditta di competenza per il trasporto. Durante il tempo che trascorre tra la richiesta e il momento del ritiro, l'utente invia il pagamento in TransportAsset al contract account che manterrà i fondi inviati. Il contratto consentirà il trasferimento di tali risorse verso l'addetto al trasporto solo nel caso in cui esso sia in possesso di un codice di verifica segreto fornito fisicamente dall'utente al momento del ritiro. Nel caso in cui il ritiro non avvenga entro un limite di tempo precedentemente definito, l'utente ha la possibilità di richiedere i fondi depositati nel contract account.
I diagrammi illustrati nelle figure 4.6 e 4.7 sono relativi ai moduli coinvolti nella gestione del caso d'uso rispettivamente per l'addetto al trasporto(`Receiver`) e per l'utente(`Owner`). Il processo di interazione tra i diversi moduli coinvolti è analogo per entrambi gli attori. Descriviamo tale processo dal punto di vista dell'addetto al trasporto. Il risultato della compilazione di `EscrowContract`, ovvero, `program`, viene ricevuto da `TransactionsManager`. L'addetto al trasporto invoca `TransactionsManager` mediante il metodo `receiver_claim()`. A questo punto viene generata la transazione `txn` e viene richiesta a `TransactionsClass` la firma logica con la quale autorizzarla(`lsig`). Per l'autorizzazione di `txn` viene ancora una volta invocato `TransactionsClass` mediante il metodo `LogicSigTransaction(txn, lsig)`, e viene restituita la transazione firmata(`stxn`). Per concludere, `Node` invia la transazione ad `AlgorandBlockchain` dopo la sua invocazione da parte di `TransactionsManager` mediante il metodo `send_transactions(stxn)`. L'unica differenza con il diagramma di comunicazione relativo all'utente, consiste nel nome del metodo che quest'ultimo utilizza per l'invocazione di `TransactionsManager`, che nel diagramma in Fig. 4.7 prende il nome di `owner_claim()`.

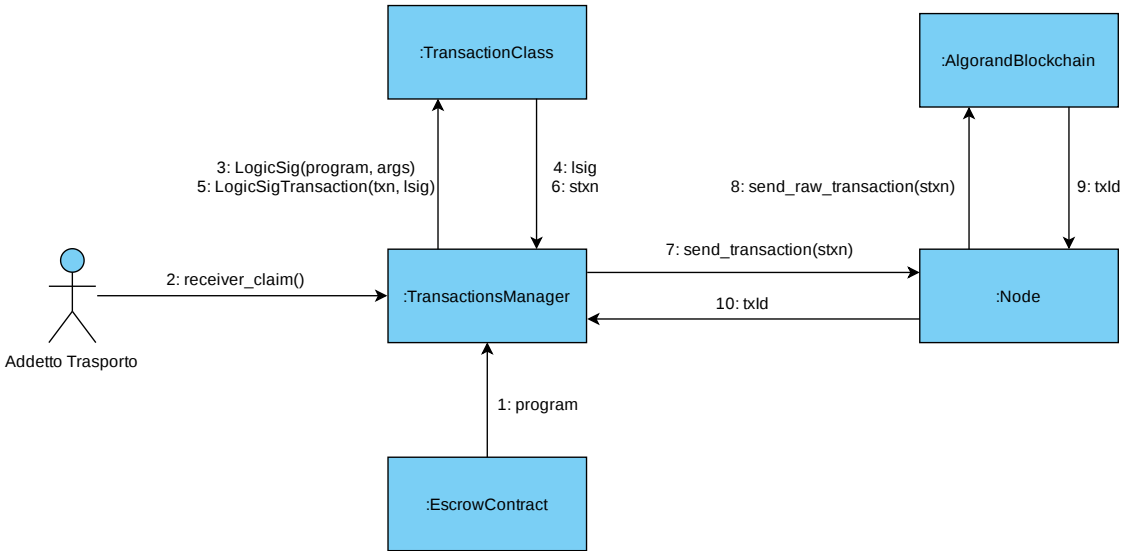

Figura 4.6. Diagramma di comunicazione Use Case 5 per il Receiver

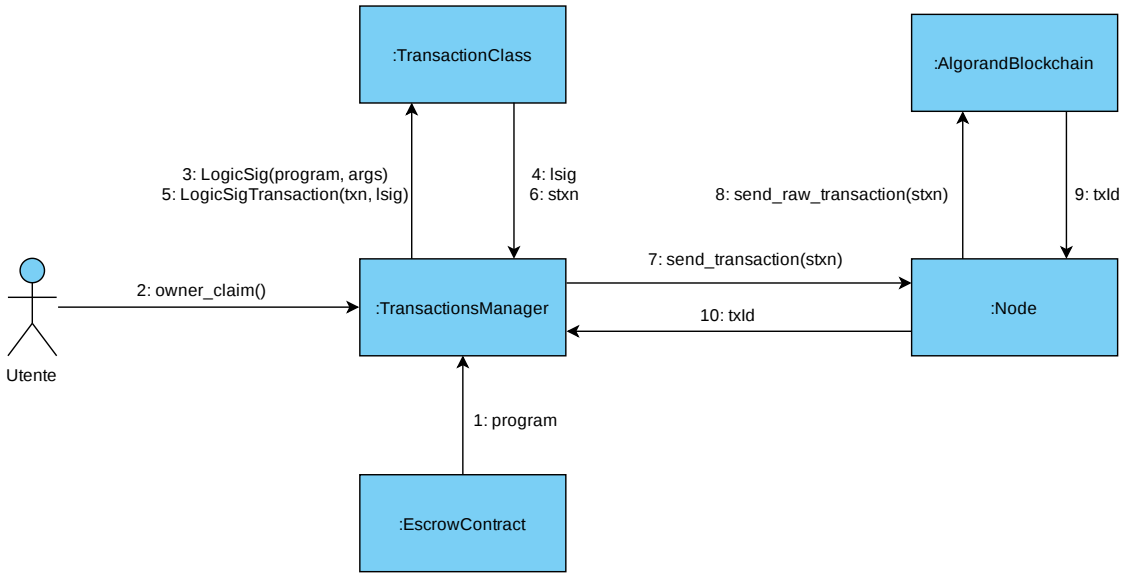

Figura 4.7. Diagramma di comunicazione Use Case 5 per l'Owner

Capitolo 5

Implementazione e valutazione

In questa capitolo verranno analizzati nel dettaglio i diversi smart contract implementati per la realizzazione dei casi d'uso appena illustrati.
Come già ampiamente discusso nella sezione 3.4, TEAL è il linguaggio con cui vengono implementati gli Algorand Smart Contract. Per l'implementazione del caso di studio esposto nella sezione precedente, oltre a TEAL, è stato utilizzato anche il linguaggio Python. Python è tra i linguaggi di cui Algorand consente l'utilizzo per l'implementazione del lato client al fine di interagire con la sua blockchain. Oltre all'interazione con la blockchain, Python è stato adoperato anche per semplificare il processo di programmazione degli smart contract.
Date le difficoltà spesso riscontrate nella realizzazione di contratti codificati in TEAL, a causa della sua natura di linguaggio a stack, è stata sviluppata una libreria completamente open-source dal nome PyTeal [17]. PyTeal consente di scrivere in linguaggio Python smart contract di tipo stateless e stateful e di compilarli successivamente in codice TEAL. È importante però specificare che la compilazione effettuata mediante PyTeal non restituisce un byte code, bensì, una semplice traduzione del contratto in TEAL. La compilazione effettiva può essere effettuata mediante l'uso di linguaggi come Python, Go, Javascript, Java, oppure mediante l'interfaccia a linea di comando Goal.
Nelle sottosezioni che seguono esporremo, per ogni caso d'uso, una breve descrizione relativa alla sua implementazione. Verranno inoltre mostrati diversi frammenti di codice di smart contract scritti attraverso la libreria PyTeal, poiché risulta essere di più agevole comprensione. È possibile visionare interamente tutti gli smart contract implementati in linguaggio TEAL al seguente link `https://github.com/rafmc98/Studio-e-realizzazione-di-smart-contract-su-piattaforma-Algorand`. Tutti gli script relativi all'implementazione del caso di studio descritti sono stati eseguiti all'interno della così detta rete Testnet, ovvero, una rete che rispecchia in termini di software e prestazione la rete principale (Mainnet) ma utilizza Algo di prova

distribuiti mediante un faucet.

5.1 Caso d'uso 1: Riciclare rifiuti ingombranti

Per la realizzazione di questo caso d'uso sono stati implementati un contratto stateless di tipo contract account che prende il nome di "recycleStateless.teal", ed un contratto stateful, rappresentato dalla coppia di file "approval.teal" e "clear_state.teal". Inoltre per quanto riguarda l'implementazione di questo caso d'uso il ruolo dell'oggetto `TransactionsManager` viene assunto dal file "create_transactions.py".
Lo smart contract stateless, come mostrato nella Fig. 5.1, verifica che il codice segreto ricevuto come argomento da parte dell'operatore sia corretto(`sec_control`) e che il premio stabilito in EcoAsset sia conforme alla fase di valutazione effettuata al momento del deposito(`amount_controls`).

```
# conditions to check if the assetAmount is correct
amount_controls = Or(
    And(Btoi(Arg(1)) >= Int(0), Btoi(Arg(1)) <= Int(10), Gtxn[0].asset_amount() == Int(1) * Btoi(Arg(2))),
    And(Btoi(Arg(1)) >= Int(11), Btoi(Arg(1)) <= Int(50), Gtxn[0].asset_amount() == Int(2) * Btoi(Arg(2))),
    And(Btoi(Arg(1)) >= Int(51), Btoi(Arg(1)) <= Int(100), Gtxn[0].asset_amount() == Int(3) * Btoi(Arg(2))),
    And(Btoi(Arg(1)) >= Int(101), Btoi(Arg(1)) <= Int(200), Gtxn[0].asset_amount() == Int(4) * Btoi(Arg(2))),
    And(Btoi(Arg(1)) >= Int(201), Gtxn[0].asset_amount() == Int(5) * Btoi(Arg(2)))
)

# conditions to check if the security code is correct
sec_control = Or(
    Arg(0) == secret_code[0],
    Arg(0) == secret_code[1],
    Arg(0) == secret_code[2],
    Arg(0) == secret_code[3]
)
```

Figura 5.1. Frammento codice PyTeal Smart Contract Stateless

Nel frammento di codice illustrato nella figura Fig. 5.2 vengono mostrate le operazione effettuate dallo smart contract stateful. Queste operazioni vengono attivati in base agli argomenti che il contratto riceve. Al momento della creazione del contratto, ovvero, quando il campo `ApplicationId` corrisponde a `0`, vengono inizializzati i contatori relativi al numero totale di ricicli effettuati dagli utenti e ai diversi livelli di riciclabilità(`on_creation`). Quando viene ricevuta come argomento la stringa `'update_recycle_counter'`, viene attivata la sezione `on_increment` incaricata di aggiornare i contatori relativi al totale di ricicli effettuati e al livello di riciclabilità passato come argomento.

```
# get user counter
get_value_of_user = App.localGetEx(Int(0), App.id(), Bytes("recycle_counter"))

# get total recycle counter
get_total_recycle = App.globalGet(Bytes("tot_recycle_counter"))

# execute when Txn.applicationId() = 0
on_creation = Seq([
    App.globalPut(Bytes("Creator"), Txn.sender()),
    App.globalPut(Bytes("tot_recycle_counter"), Int(0)),
    App.globalPut(Bytes("recyclibility_level_1"), Int(0)),
    App.globalPut(Bytes("recyclibility_level_2"), Int(0)),
    App.globalPut(Bytes("recyclibility_level_3"), Int(0)),
    App.globalPut(Bytes("recyclibility_level_4"), Int(0)),
    App.globalPut(Bytes("recyclibility_level_5"), Int(0)),
    Return(Int(1))
])

    # execute when Txn.application_args[0] = 'update_recycle_counter'
on_increment = Seq([
    get_value_of_user,
    App.globalPut(Bytes("tot_recycle_counter"), get_total_recycle + Int(1)),
    If(Txn.application_args[0] == Bytes("1"), App.globalPut(Bytes("recyclibility_level_1"), get_level_1 + Int(1))),
    If(Txn.application_args[0] == Bytes("2"), App.globalPut(Bytes("recyclibility_level_2"), get_level_2 + Int(1))),
    If(Txn.application_args[0] == Bytes("3"), App.globalPut(Bytes("recyclibility_level_3"), get_level_3 + Int(1))),
    If(Txn.application_args[0] == Bytes("4"), App.globalPut(Bytes("recyclibility_level_4"), get_level_4 + Int(1))),
    If(Txn.application_args[0] == Bytes("5"), App.globalPut(Bytes("recyclibility_level_5"), get_level_5 + Int(1))),
    App.localPut(Int(0), Bytes("recycle_counter"), get_value_of_user.value() + Int(1)),
    Return(Int(1))
])
```

Figura 5.2. Frammento codice PyTeal Smart Contract Stateful

5.2 Caso d'uso 2: Acquistare MetroAsset

Il secondo caso d'uso esposto implementa uno scambio di Algorand Standard Asset realizzato mediante l'uso di un atomic transfer. Per questo scenario è stato generato un contratto stateless("assetExchange.teal") che verifica le condizioni dello scambio tra ECO e METRO. La creazione delle transazioni che costituiscono il trasferimento atomico è stata affidata allo script contenuto nel file "groupTransactions.py". I controlli di maggior rilevanza che vengono effettuati sono illustrati nella Fig. 5.3. Il contratto si occupa di verificare che gli asset coinvolti nello scambio siano effettivamente quelli corretti, che il destinatario della transazione per l'invio di MetroAsset sia anche il mittente della transazione per l'invio di EcoAsset, e infine che il tasso di scambio tra ECO e METRO venga rispettato.

```
# conditions to check the asset exchange transactions fields
controls = And(
    Gtxn[0].asset_receiver() == Gtxn[1].sender(),

    Gtxn[0].type_enum() == TxnType.AssetTransfer,
    # <ID MetroAsset>
    Gtxn[0].xfer_asset() == Int(14638139),
    Gtxn[0].asset_amount() == Int(1),
    Gtxn[0].fee() <= Int(1000),
    Gtxn[0].asset_close_to() == Global.zero_address(),
    Gtxn[0].asset_sender() == Global.zero_address(),

    Gtxn[1].type_enum() == TxnType.AssetTransfer,
    # <ID EcoAsset>
    Gtxn[1].xfer_asset() == Int(14531028),
    Gtxn[1].asset_amount() == Int(15),
    Gtxn[1].fee() <= Int(1000),
    Gtxn[1].asset_close_to() == Global.zero_address(),
    # <Addr IsolaEcologica>
    Gtxn[1].asset_receiver() == Addr("ROV6LJIUDCSQEVX2AU7CWGOEZT2DQJUDLOYUP56MQGKVI2ECZXNUSUDOLU"),
    Gtxn[1].asset_sender() == Global.zero_address(),
)
```

Figura 5.3. Frammento codice PyTeal Smart Contract Stateless

5.3 Caso d'uso 3: Acquistare EcoAsset

La realizzazione di questo caso d'uso risulta essere molto simile al precedente e coinvolge anche qui un solo contratto di tipo stateless rappresentato nella progettazione come `ExchangeAlgoContract`. Tale contratto nella fase di implementazione prende il nome di "exchangeAlgo.teal", mentre l'oggetto `TransactionsManager` quello di "exchangeTransaction.py". Le verifiche effettuate dal contratto stateless sono le medesime descritte nella sottosezione precedente, in particolare in questo caso, poiché il tasso di scambio è variabile, viene ricalcolato il totale di unità di EcoAsset che vengono inviate.

5.4 Caso d'uso 4: Acquistare TransportAsset

Anche per questo caso di studio i controlli risultato essere molto simili a quelli descritti nella sottosezione 5.2. La differenza è nella presenza del contratto stateful, citato al punto 5.1, rappresentato dalla coppia di file "approval.teal" e "clear_state.teal". Anche in questo contesto viene realizzato un contratto stateless implementato mediante il file "transportStateless.teal". Il gestore delle transazioni, rappresentato dall'oggetto `TransactionsManager` nella la fase di progettazione, prende il nome di "transportExchange.py" nella fase implementativa.
Lo smart contract stateful avvia un operazione per il controllo del numero di depositi effettuati dall'utente(`on_check`) quando riceve la stringa `'check_user_counter'` come argomento(Fig. 5.4).

```
# execute when Txn.application_args[1] = 'check_user_counter'
on_check = Seq([
    get_value_of_user,
    If(get_value_of_user.value() >= Int(10),
        Return(Int(1))
    ),
    Return(Int(0))
])
```

Figura 5.4. Frammento codice PyTeal Smart Contract Stateful

Il contratto stateless, come illustrato nella Fig. 5.5, effettua controlli sull'Application Call, verificando che l'`ApplicationId` sia corretto e che venga passata come argomento la stringa `'check_user_value'`.

```
# Application call to check the user value
Gtxn[2].type_enum() == TxnType.ApplicationCall,
Gtxn[2].sender() == Gtxn[0].asset_receiver(),
Gtxn[2].application_id() == Int(14877441),
Gtxn[2].application_args[0] == Bytes("check_user_value")
```

Figura 5.5. Frammento codice PyTeal Smart Contract Stateless

5.5 Caso d'uso 5: Richiedere ritiro ingombranti a domicilio

Per la realizzazione di quest'ultimo caso d'uso è stato definito un singolo smart contract di tipo stateless che implementa un fondo di deposito a garanzia(escrow). Questo contratto è rappresentato dal file "escrow.teal". L'oggetto avente il ruolo di gestore delle transazioni nella fase di progettazione, ovvero, `TransactionsManager` viene assunto dai due file "owner_claim.py" e "receiver_claim()". Il contratto citato ha l'incarico di mantenere le risorse che gli vengono inviate e di rilasciarle a chi ne ha diritto solo al verificarsi di alcune condizioni, la cui codifica è mostrata nella Fig. 5.6.
Le condizioni relative all'addetto al trasporto degli ingombranti sono definite in `receiver_cond` e verificano che l'indirizzo dell'account ricevente sia corretto e che venga fornita la giusta chiave di sicurezza per lo sblocco dei fondi. Infine le condizioni relative all'utente sono definite in `owner_cond` e verificano che l'indirizzo dell'account ricevente sia corretto e che il tempo massimo per il ritiro a domicilio sia scaduto, se così fosse l'utente può recuperare i fondi versati in precedenza.

```
# receiver conditions to closeOut
receiver_cond = And(
    Txn.receiver() == tmpl_receiver,
    tmpl_hash_fn(Arg(0)) == tmpl_secret
)

# owner conditions to closeOut
owner_cond = And(
    Txn.receiver() == tmpl_owner,
    Txn.first_valid() > Int(tmpl_timeout)
)
```

Figura 5.6. Frammento codice PyTeal Smart Contract Stateless

5.6 Valutazione dei costi e dei tempi

Per concludere questo capitolo abbiamo deciso di illustrare una breve panoramica dei costi di un'eventuale realizzazione effettiva del caso di studio proposto in termini commissioni in Algo.
L'assenza di risoluzione di puzzle crittografici, e la conseguente assenza di miners, fanno si che Algorand riesca a garantire costi di commissione per singola transazione pari a 0.001 Algo. La valutazione dei costi dello scenario proposto nel Cap. 4 è stata realizzata considerando il punto di vista del fornitore di servizi, ovvero, una società urbanistica di smaltimento rifiuti.
Il sistema di cui vogliamo fare una valutazione in termini di costi necessita di alcune configurazioni iniziali che coinvolgono ovviamente diverse transazioni.
La creazione e la configurazione degli asset coinvolti nel caso di studio comportano costi di commissione:

- **EcoAsset**: Coinvolge tre transazioni, una per la sua creazione e due per la configurazione dei suoi parametri mutabili, inoltre richiede che il bilancio dell'account creatore venga incrementato di 0.001 Algo:

$$(0.001 Algo \cdot 3) + 0.001 Algo = 0.004 Algo$$

- **MetroAsset**: Coinvolge una sola transazione relativa alla sua creazione, e richiede un incremento di 0.001 del bilancio dell'account creatore:

$$0.001 Algo + 0.001 Algo = 0.002 Algo$$

- **TransportAsset**: Coinvolge una sola transazione relativa alla sua creazione,

e richiede un incremento di 0.001 del bilancio dell'account creatore:

$$0.001Algo + 0.001Algo = 0.002Algo$$

È richiesto inoltre ai singoli utenti, che fanno opt-in degli asset appena citati, un incremento di 0.001 Algo del loro saldo minimo.
Anche la creazione dei diversi smart contract coinvolti necessitano di alcune transazioni relative alla loro creazione e alla loro configurazione:

- **Smart Contract Stateless**: Gli smart contract stateless necessari alla realizzazione del caso di studio proposto sono 5, sebbene la loro compilazione non comporti costi di commissione, questi necessitano di un saldo minimo pari a 0.001 Algo per la loro attivazione. I contratti ricevono tale saldo mediante diverse transazioni ognuna con le proprie commissioni. Inoltre 4 dei 5 contract account implementati necessitano l'invio di transazioni di tipo opt-in per l'aggiunta di un asset al proprio wallet. Considerando le transazioni appena descritte il calcolo delle commissioni che ne consegue è il seguente:

  $$(0.001Algo + 0.001Algo) \cdot 5 + 0.001Algo \cdot 4 = 0.014Algo$$

  Inoltre il valore appena ottenuto necessita di essere incrementato in relazione a quante transazioni vengono utilizzate per fornire le disponibilità di asset agli smart contract che includono l'operazione di opt-in all'interno della loro logica.

- **Smart Contract Stateful**: Nel caso di studio illustrato è coinvolto un solo contratto di tipo stateful. Questa tipologia di contratti ha un calcolo dei costi variabile in base alla sua configurazione. Anche la creazione di contratti stateful necessita di un saldo minino per l'account creator e per gli account che ne fanno opt-in. In particolare viene utilizzato il calcolo seguente per la definizione di tale saldo minimo richiesto(valori espressi in termini di microAlgo):

  $$100000 + (25000 + 3500) \cdot schema.NumUint+$$
  $$+(25000 + 25000) \cdot schema.NumByteSlice$$

  100000 microAlgo per la creazione e per l'opt-in, 25000 microAlgo per l'occupazione di uno spazio di memoria, sommato a 3500 microAlgo se di tipo intero, 25000 microAlgo se di tipo Byte, il tutto moltiplicato per la quantità di tali spazi, ovvero, schema.NumUint per gli interi e schema.NumByteSlice per i Byte. Nel nostro caso lo stato globale coinvolge 1 spazio di memoria di tipo Byte per memorizzare l'indirizzo dell'account creatore, 6 spazi di memoria di

tipo intero per il totale di depositi effettuati e per i contatori dei 5 diversi livelli di riciclabilità. Lo stato locale infine è composto da uno spazio di tipo intero per la memorizzazione del totale delle consegne effettuate dai singoli utenti. È importante dire che i costi relativi allo stato globale sono responsabilità dell'account creator, mentre quelli dello stato locale dell'utente che esegue l'opt-in del contratto. Il saldo minimo relativo al fornitore del servizio, ovvero, l'account creatore, viene incrementato del risultato del seguente calcolo:

$$100000 + (25000 + 3500) \cdot 6 + (25000 + 25000) = 321000 microAlgo = 0.321 Algo$$

Per quanto concerne gli utenti che fanno opt-in del contratto, viene richiesto un incremento del saldo minimo pari a:

$$100000 + (25000 + 3500) = 128000 microAlgo = 0.1285 Algo$$

Nell'implementazione degli scenari proposti non sono stati considerati eventuali rimborsi all'utente relativi ai costi di commissione poiché non rilevante al fine dello studio illustrato in questo elaborato. Nonostante ciò per concludere la valutazione dei costi consideriamo anche l'invio di transazioni di rimborso verso gli utenti che usufruiscono del servizio. Riassumendo il totale dei costi di configurazione iniziale sono i seguenti:

- **Costi configurazione asset**:

  $$0.004 Algo + 0.002 Algo + 0.002 Algo = 0.008 Algo$$

  Viene richiesto un incremento di 0.002 Algo per ogni account che esegue l'opt-in di un asset.

- **Costi creazione smart contract**:

  $$0.014 Algo + 0.321 Algo = 0.335 Algo$$

  Viene richiesto un incremento di 0.1285 Algo per ogni account che esegue l'opt-in del contratto stateful.

Riassumendo si ottiene un totale di 0.343 Algo (non considerando le operazioni di opt-in), che stando all'attuale valore di Algo in termini di euro [12], equivale a 0.39445€. Infine i risultati sono mostrati nella tabella seguente e illustrano i costi che il fornitore del servizio paga per l'esecuzione di ogni singolo caso d'uso considerando anche le transazioni per il rimborso dei costi di commissione agli utenti.

Id	Titolo	Commissioni fornitore
1	Ricicla Ingombranti	0.003 Algo
2	Acquista METRO	0.003 Algo
3	Acquista ECO	0.003 Algo
4	Acquista TRAN	0.004 Algo
5	Ritiro a Domicilio	0.003 Algo

Considerando un determinato numero di utenti che aderiscono a questo sistema e l'utilizzo che questi fanno di ognuno dei casi d'uso proposti, è possibile calcolare una stima dei costi totali relativi al fornitore del servizio su base giornaliera, settimanale o mensile.
Di norma Algorand consente l'esecuzione di oltre 1000 transazioni al secondo ad una velocità per l'approvazione di nuovi blocchi pari circa a 5 secondi. Inoltre quanto appena detto vale non solo per le singole transazioni ma anche per quelle coinvolte nei trasferimenti atomici. Per constatare queste affermazioni abbiamo effettuato un analisi delle prestazioni espresse in termini di tempi di approvazione delle transazioni. La valutazione espressa in questi termini è stata realizzata eseguendo gli script mediante l'uso della rete Testnet. Questa analisi comprende il calcolo dei tempi di finalità di ogni singolo caso d'uso a partire dal momento di invio delle transazioni fino all'istante in cui viene assegnato un `confirmed-round` a queste ultime, come illustrato anche nella figura seguente.

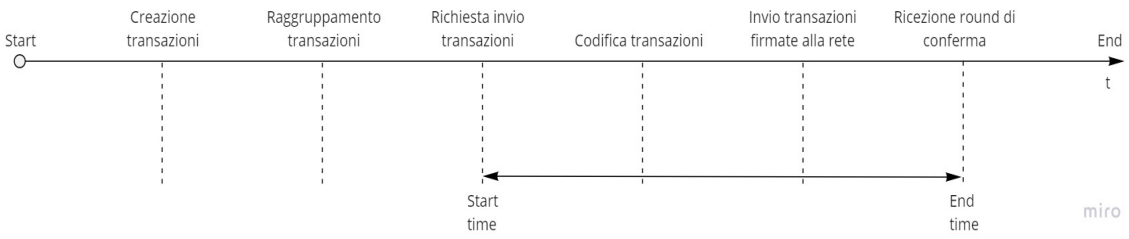

Figura 5.7. Linea temporale di invio transazioni

Ogni script relativo al `TransactionManager` dei diversi casi d'uso è stato eseguito esattamente dieci volte ed ogni esecuzione ha restituito il tempo di finalità da esso derivato. Dalle tempistiche ottenute si è analizzato nello specifico il tempo minimo, massimo e medio. Per quanto concerne il caso d'uso relativo al ritiro di ingombranti a domicilio si è scelto di analizzare solamente le transazioni relative all'Owner in quanto sia la transazione diretta all'utente, sia la transazione diretta all'addetto al trasporto, possiedono la medesima struttura e non presentano alcuna differenza dal

punto di vista logico. Nella tabella che segue sono riassunti i dati ottenuti dalla valutazione effettuata.

Id	**Titolo**	**Tempo Finalità Transazioni**		
		Minimo	**Medio**	**Massimo**
1	Ricicla Ingombranti	4.795 s	5.962 s	8.397 s
2	Acquista METRO	5.060 s	6.763 s	8.142 s
3	Acquista ECO	4.562 s	6.740 s	8.770 s
4	Acquista TRAN	4.726 s	6.521 s	8.654 s
5	Ritiro a domicilio	4.889 s	6.329 s	8.083 s
Medie Totali		4.806 s	6.463 s	8.409 s

Come possiamo notare dai dati ottenuti durante il calcolo dei tempi di approvazione delle transazioni, notiamo che questi sono coerenti e compatibili tra loro, infatti, è facile vedere che tutti i tempi minimi riscontrati per ogni caso d'uso oscillano tra i 4 e i 5 secondi, i tempi medi tra i 5 e i 6 secondi, mentre i tempi massimi intorno agli 8 secondi. Sono stati riscontrati alcuni calcoli sospetti dovuti a tempi che si aggiravano sugli 8 secondi, spesso il doppio dei tempi minimi, ma ciò è stato considerato come un'anomalia accettabile poiché coerente con tutte le valutazioni effettuate. Inoltre i calcoli sono stati effettuati analogamente anche in date diverse tra loro e in momenti differenti della giornata in modo tale da avere una visione più ampia della valutazione effettuata. Nonostante ciò non è si è riscontrato alcun dato incompatibile con quelli precedentemente ottenuti.
Algorand ha inoltre da poco introdotto una dashboard attraverso la quale è possibile monitorare l'attuale scalabilità, velocità, decentralizzazione e adozione del protocollo open source Algorand [5]. La dashboard fornisce una prospettiva in tempo reale sulle prestazioni di Algorand e sul suo crescente utilizzo. Vengono monitorati ad esempio il numero di transazioni inviate al secondo, il tempo di finalità dei blocchi, e molti altri aspetti rilevanti nell'ambito blockchain. Le misurazioni a disposizione sulla dashboard sono effettuate sulla rete principale di Algorand, ovvero, la Mainnet. Osservando tali dati è possibile notare che il tempo di approvazione per ogni blocco difficilmente supera i 4.5 secondi e questo anche considerando un largo periodo di tempo come 90 giorni. Da questa osservazione possiamo affermare che la Testnet, ovvero, la rete su cui abbiamo effettuato la valutazione, offre prestazioni meno elevate se messe a confronto con quelle garantite dalla Mainnet, sebbene siano più che accettabili per una rete dedita alla fase di sviluppo. In conclusione possiamo dire che Algorand offre una buona solidità in termini di tempi di approvazione senza mai restituire tempi di attesa sconvenienti, ed inoltre garantisce performance elevate anche per la sua rete di testing soprattutto se messe a confronto con quelle offerte

da blockchain più diffuse come Bitcoin o Ethereum.

Capitolo 6

Conclusioni

In questo capitolo ripercorreremo i punti di maggior interesse e le conclusioni emerse durante la stesura dell'elaborato. Illustreremo un breve riepilogo di ogni capitolo evidenziando gli aspetti più rilevanti di ognuno di questi, sia da un punto di vista architetturale sia da un punto di vista progettuale, ovvero, relativo all'economia circolare e al caso di studio descritto nel capitolo precedente. Per concludere mostreremo alcuni di quelli che potrebbero essere gli sviluppi futuri del caso di studio realizzato.

6.1 Implicazioni e sviluppi futuri

La stesura di questo elaborato, insieme al lavoro progettuale svolto, ha consentito di comprendere le potenzialità delle nuove tecnologie legate al mondo della finanza decentralizzata. Possiamo affermare che quest'ultima offre grandi possibilità di sviluppo garantendo agli utenti il pieno controllo delle loro risorse. Abbiamo visto come tutto ciò sia possibile soprattutto grazie all'introduzione degli Smart Contract. Tra le ormai diverse tecnologie DeFi abbiamo analizzato in particolare la blockchain Algorand e soprattutto come questa ha scelto di affrontare le sfide imposte dal mondo della finanza decentralizzata. Seppur Algorand sia ancora in piena fase di sviluppo ha già un'architettura in grado di garantire numerosi vantaggi. Abbiamo visto come il Pure Proof of Stake sia in grado di offrire un'alta scalabilità, sicurezza e soprattutto decentralizzazione. Gli Algorand Standard Asset consentono a tutti gli utenti di rappresentare e personalizzare risorse, e come abbiamo potuto constatare le loro possibilità di utilizzo sono molteplici. Questi consentono di mettere in pratica controlli opzionali al fine di garantire requisiti aziendali e normativi, come ad esempio il loro trasferimento forzato, la creazione di liste privilegiate per il loro utilizzo, o l'imposizione di limiti sulla loro diffusione per indirizzi specifici. In particolare nel nostro caso di studio sono stati utilizzati al fine di rappresentare risorse scambiabili

per la fruizione di servizi pubblici. Inoltre su queste risorse sono stati imposte limitazione in modo da controllare e regolamentare la loro diffusione.
Un altro importante aspetto di cui lo studio esposto ha beneficiato sono stati i trasferimenti atomici, mediante i quali è stato possibile realizzare la maggior parte dei casi d'uso progettati al fine di implementare scambi circolari e decentralizzati garantendo sicurezza e affidabilità.
Ovviamente il ruolo di maggior rilevanza è stato assunto dagli Algorand Smart Contract(ASC). L'utilizzo di questa nuova tecnologia ha consentito l'implementazione dei diversi scenari proposti. Mediante gli ASC è stato possibile sviluppare scambi decentralizzati di asset garantendone la corretta finalità mediante l'imposizione di specifiche clausole. Inoltre hanno consentito la realizzazione di un contratto rappresentante un fondo di deposito a garanzia, ovvero, uno dei casi d'uso più conosciuti nell'ambito della finanza decentralizzata. Infine una delle maggiori novità che gli Algorand Smart Contract hanno apportato nel mondo della DeFi sono stati gli Stateful Smart Contract, contratti che offrono la possibilità di memorizzare uno stato globale e locale on-chain garantendo l'immutabilità di tali informazioni. Nel nostro caso specifico questa tipologia di contratto è stata utilizzata al fine di mantenere informazioni riguardanti il riciclo di materiali ingombranti a fini statistici e non solo.
Inoltre è stata da poco rilasciata una nuova versione del software Algorand costituita dall'introduzione della Algorand Virtual Machine (AVM). la AVM è in grado di supportare un linguaggio Turing-completo, il quale consente agli sviluppatori di implementare cicli e subroutine, oltre ad offrire la possibilità di condivisione di dati tra differenti chiamate di contratto quando combinate mediante trasferimenti atomici.
Gli sviluppatori che si lanciano nel mondo della finanza decentralizzata sono in numero sempre maggiore. Vengono create applicazioni decentralizzate più o meno sofisticate, partendo da semplici transazioni per pagamenti peer to peer fino ad arrivare ad applicazioni di maggiore complessità. Gli Algorand Smart Contract offrono a tutti gli sviluppatori grandi possibilità di utilizzo, grazie alla loro alta scalabilità, velocità e sicurezza, caratteristiche che consentiranno ad Algorand di assumere un ruolo chiave nel futuro della DeFi.
La presenza di rifiuti ingombranti nelle strade rimane ad oggi un problema ancora molto diffuso soprattutto nelle grandi metropoli. Questo è uno dei motivi principali per cui le istituzioni dovrebbero mettere in atto progetti simili a quello proposto, in modo da rendere coscienti e attivi i cittadini verso argomenti come il riciclaggio e il riutilizzo. Oltre agli aspetti descritti, il sistema proposto potrebbe subire sviluppi futuri e aprirsi verso un maggior numero di settori differenti. Alcune delle possibilità di sviluppo sono descritte qui di seguito.

- Sistema di segnalazione della presenza di rifiuti ingombranti nelle strade.
- Acquisto di biglietti per eventi culturali quali, mostre, spettacoli teatrali, cinema e musei, mediante compensi in asset.
- Compensi in asset ricavati dal riciclaggio di rifiuti meno ingombranti come bottiglie in plastica o carta.
- Compensi in asset per l'utilizzo di sistemi di trasporto non inquinanti e non motorizzati all'interno delle aree metropolitane, quali ad esempio la bicicletta.

Mediante anche l'implementazione delle opzioni appena elencate si potrà fornire un nuovo punto di vista sulle tematiche ambientali e sulla blockchain. Inoltre tali progetti consentiranno alle nuove tecnologie di essere facilmente accessibili e fruibili da un numero sempre maggiore di persone, agevolando soprattutto la loro quotidianità.

Bibliografia

[1] Saif Ahmed Abdulhakeem, Qiuling Hu, et al. Powered by blockchain technology, defi (decentralized finance) strives to increase financial inclusion of the unbanked by reshaping the world financial system. *Modern Economy*, 12(01):1, 2021.

[2] Algorand. Next generation smart contracts for the defi world, 2019. URL https://www.algorand.com/smart-contracts-defi. Consultato il 27 Aprile 2021.

[3] Algorand. Smart contracts, 2019. URL https://www.algorand.com/technology#ASC1. Consultato il 26 Aprile 2021.

[4] Algorand. Developer portal, 2019. URL https://developer.algorand.org. Consultato il 04 Maggio 2021.

[5] Algorand. Algorand mainet metrics dashboard, 2021. URL https://metrics.algorand.org. Consultato il 17 giugno 2021.

[6] Algorand Foundation. Algorand protocol, 2019. URL https://algorand.foundation/algorand-protocol. Consultato il 18 Aprile 2021.

[7] Algorand Foundation. Understanding technicalities behind the pure proof-of-stake protocol algorand uses, 2019. URL https://community.algorand.org/blog/understanding-the-technicalities-behind-the-pure-proof-of-stake-protocol-algorand-uses. Consultato il 19 Aprile 2021.

[8] Algorand Foundation. Algorand's smart contract architecture, 2020. URL https://www.algorand.com/resources/blog/algorand-smart-contract-architecture. Consultato il 20 Aprile 2020.

[9] Brocardi. Cosa fungibile, 2019. URL https://www.brocardi.it/dizionario/1564.html. Consultato il 10 Maggio 2021.

[10] Vitalik Buterin et al. Ethereum white paper. *GitHub repository*, 1:22–23, 2013.

[11] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. doi: 10.1016/j.tcs.2019.02.001. URL https://doi.org/10.1016/j.tcs.2019.02.001.

[12] CoinMarketCap. Algorad(algo), 2019. URL https://coinmarketcap.com/it/currencies/algorand/. Consultato il 27 Aprile 2021.

[13] Haardik. Understanding algorand — the blockchain which claims to solve the trilemma, 2019. URL https://medium.com/@haardikk21/understanding-algorand-the-blockchain-which-claims-to-solve-the-trilemma-56f4c3ef0e81. Consultato il 19 Aprile 2021.

[14] Robby Houben and Alexander Snyers. Cryptocurrencies and blockchain. *Bruxelles: European Parliament*, 2018.

[15] Liz Baran. Algorand standard assets, 2019. URL https://medium.com/algorand/algorand-standard-assets-efda8afcfc0a. Consultato il 25 Aprile 2021.

[16] PlanetWatch. Planetwatch company, 2020. URL https://planetwatch.io/company/. Consultato il 25 Aprile 2021.

[17] PyTeal. Pyteal, 2020. URL https://pyteal.readthedocs.io/en/latest/overview.html. Consultato l'11 Maggio 2021.

[18] Fabian Schär. Decentralized finance: On blockchain-and smart contract-based financial markets. *FRB of St. Louis Review*, 2021.

[19] Wikipedia contributors. Circular economy — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/w/index.php?title=Circular_economy&oldid=1031238553. Consultato il 21 Aprile.