



UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA TRIENNALE IN INFORMATICA

Analisi della blockchain Algorand mediante porting dell'applicazione Cryptokitties

Relatori:

Prof.ssa Laura Ricci

Dott. Andrea Lisi

Candidato:

Pietro Scarso

Anno Accademico 2019/2020

Indice

1	Introduzione	2
1.1	Scopi e motivazioni	4
1.2	Struttura del documento	6
2	Stato dell'arte	7
2.1	Fondamenti di crittografia	7
2.2	Blockchain	10
2.2.1	PoW vs PoS	13
2.2.2	Delegated vs Bonded PoS	13
2.2.3	Blockchain forks	15
2.3	Evoluzione delle Blockchain	17
2.4	Fungible & Non-Fungible tokens	19
3	Algorand	21
3.1	Blockchain Trilemma	21
3.1.1	Pure Proof-of-Stake	22
3.1.2	PPoS: Analisi della sicurezza	23
3.1.3	Consensus Algorithm	24
3.2	Architettura di rete	26
3.3	Algorand Smart Contracts (ASC1)	28
3.3.1	TEAL: Transaction Execution Approval Language	29
3.3.2	Stateful Smart Contracts	30
3.3.3	Stateless Smart Contracts	31
3.4	Algorand Standard Assets (ASA)	33
3.4.1	Fungible & Non-Fungible	34

3.5	Transazioni	35
3.5.1	Struttura e tipologie	36
3.5.2	Atomic Transfers	46
3.6	Indexer V2	48
4	AlgoKitties	50
4.1	Ethereum Cryptokitties	50
4.2	Implementazione su Algorand	52
4.2.1	Strumenti di sviluppo	53
4.2.2	Scelte implementative	54
4.2.3	Server Application	56
4.2.4	Client Application	60
5	Analisi e valutazione	67
5.1	Modello di ricompense	67
5.2	Analisi dei costi	68
5.3	Tempi di conferma	70
6	Conclusioni	74
	Riferimenti	76

Sommario

L'obiettivo del tirocinio è quello di analizzare le caratteristiche distintive della blockchain Algorand, e del suo algoritmo di consenso, attraverso la riprogettazione ed implementazione dell'applicazione CryptoKitties di Ethereum. Il porting sviluppato, rinominato "Algo-Kitties", verrà utilizzato come tool di benchmark per valutare le potenzialità di Algorand e capire come questa tecnologia può essere impiegata, e quali benefici porta con sé rispetto alle alternative presenti ad oggi sul mercato.

Ringraziamenti

Giunto al termine di questo percorso sento di dover ringraziare tutti coloro che mi hanno supportato in questi anni. Comincio con il ringraziare la Prof. Laura Ricci ed il Dott. Andrea Lisi per la professionalità e la gentilezza con la quale mi hanno guidato nel mio lavoro di tirocinio. Un ringraziamento va anche al Dott. Paolo Mori ed al Dott. Andrea De Salve che sono sempre stati presenti e pronti a condividere la loro conoscenza per permettermi di svolgere al meglio il mio lavoro.

Inoltre, sono grato per tutte le persone che ho conosciuto nel corso di questa triennale, con le quali ho condiviso momenti che mi hanno fatto crescere ed hanno arricchito la mia esperienza universitaria. Tra queste un particolare ringraziamento va ad Alessandro Pignotti, che considero un caro amico oltre che collega.

Ai compagni di viaggio Alessandro Mazzearella, Jaggy e Andrea Leaf va un enorme grazie per tutte le risate ed i momenti di disagio che hanno alleviato le pene delle più intense sessioni di studio e mi hanno sempre tirato su di morale quando serviva. Un ringraziamento va anche alla mia cara amica Cristina Napolitano per esserci sempre stata sia per i momenti belli che per quelli meno belli, grazie di cuore.

Il ringraziamento più importante va alla mia famiglia per tutto il sostegno che mi avete dato in questi anni e per non aver mai smesso di credere in me.

Capitolo 1

Introduzione

L'evoluzione delle tecnologie informatiche negli ultimi decenni ha modificato in modo significativo le nostre vite, rendendo più accessibili molti servizi grazie all'impiego di sistemi software che hanno abbattuto le barriere imposte dalla distanza. Ad esempio, gli istituti bancari operano ormai tramite trasferimenti digitali permettendo di effettuare pagamenti online in maniera veloce ed efficiente. Inoltre la digitalizzazione del mercato, in aggiunta al collegamento globale per mezzo della rete internet, ha creato nuovi scenari derivati dall'integrazione e interazione tra una moltitudine di enti e/o organizzazioni diverse disperse sull'intero globo, come ad esempio gli e-shop.

Si è così creato un contesto altamente interconnesso, formato da una fitta rete di collaborazioni ed interazioni, che sta alla base della nostra quotidianità. Per permettere il collegamento tra le varie entità che contribuiscono complessivamente a fornire un servizio vi è spesso qualche entità che opera da tramite. Si pensi ad esempio agli istituti bancari che eseguono continui spostamenti di denaro per conto dei loro clienti: questi rappresentano un'autorità centrale che funge da garante tra le parti. Tuttavia, la gestione centralizzata rappresenta uno svantaggio per chi usufruisce di un determinato servizio, il quale deve sostenere dei costi derivanti dalla mediazione e deve oltretutto riporre la sua fiducia verso chi il servizio lo offre.

Questo fenomeno è descritto con il termine "*centralizzazione*", e rappresenta da sempre un problema per i sistemi informatici (e.g. circuiti di pagamento elettronici) sia in termini di costi per l'utente, che in termini di sicurezza. Tipicamente, un sistema gestito da una singola entità è soggetto ad attacchi per comprometterne il servizio, o per acquisirne i dati privati. Questo problema è noto in ambito informatico con il nome di *Single Point of Fai-*

lure. Una prima soluzione alle problematiche appena descritte è stata proposta da Satoshi Nakamoto nel 2008, il quale volle istituire una rete *Peer-to-Peer (P2P)* sulla quale gli utenti potessero effettuare scambi monetari senza la presenza di istituti di controllo centrali. Egli infatti propose il primo protocollo volto ad implementare un sistema di pagamento decentralizzato, con lo scopo di riporre nelle mani dell'intera rete il potere di autorizzare o meno una data transazione, basandosi su criteri univocamente determinati e noti a tutti. Aggiunse quindi alla rete P2P quello che viene definito un *Algoritmo di consenso*, ovvero un modo per garantire la sicurezza degli utenti impiegando regole e tecniche crittografiche, ed eliminando così la necessità di avere un garante centrale.

L'algoritmo utilizzato da Nakamoto, chiamato *Proof of Work (PoW)*[1], consta di una serie di calcoli volti alla risoluzione di puzzle crittografici complessi, che assegnano a colui che li risolve il ruolo di validatore delle transazioni. Questo procedimento, detto *mining*, permette così a ciascun utente nel sistema di validare gli eventi della rete (le transazioni) e registrarle su un libro mastro distribuito detto *ledger*. Nel complesso la tecnologia appena descritta prende il nome di *Blockchain*, e nella fattispecie quella progettata da Nakamoto è nota come *Bitcoin*. Questo approccio è stato ripreso nel seguito dalla blockchain *Ethereum*[2], la quale ha espanso il progetto iniziale con l'introduzione di *Smart Contract*, applicazioni scritte in un linguaggio Turing-completo chiamato *Solidity*, grazie alle quali è possibile realizzare applicazioni decentralizzate (le *Dapps*) che sfruttano le proprietà della tecnologia blockchain per implementare sistemi software che formano le cosiddette *Distributed Ledger Technologies (DLT)*. Le potenzialità e i campi di utilizzo delle blockchain sono così diventati innumerevoli, e vengono oggi impiegate non solo come strumento finanziario ma anche per la creazione di sistemi informatici che necessitano di trasparenza e sicurezza.

Tuttavia nel tempo sono emerse alcune problematiche legate all'uso dell'algoritmo PoW, come ad esempio l'eccessivo costo computazionale (ed energetico) affrontato per l'esecuzione dei calcoli crittografici previsti dall'algoritmo, che rendono l'impiego massivo delle blockchain spesso troppo costoso. Si è quindi creato un interesse nello studio di metodi alternativi: è il caso degli algoritmi *Proof of Stake (PoS)*, nei quali è stato sostituito l'utilizzo di computazioni complesse, con l'introduzione del concetto di "stake". Lo stake consta dei possedimenti monetari (in termini di cryptovalute) che un utente detiene. Per aggiudicarsi il ruolo di validatore, l'utente impiega il suo stake (o porzioni di esso) in forma cauzionale.

Tale approccio viene oggi utilizzato da diverse blockchain, come ad esempio *EOS.IO*¹, *Cardano*² e in una proposta di rilascio futura anche *Ethereum 2.0*³.

Ciascuna di esse implementa l'algoritmo PoS in diverse varianti, ognuna delle quali seleziona i validatori secondo criteri diversi. Abbiamo ad esempio la variante *Delegated PoS* in EOS.IO che prevede la selezione di un insieme finito di utenti ai quali viene assegnato il ruolo di commissione validatrice in base al loro stake totale, oppure ancora la variante *Bonded PoS* nella quale i validatori sono eletti sulla base della porzione di stake congelata in forma di garanzia.

Come vedremo però, anche i protocolli PoS presentano alcune debolezze: essendo l'elezione dei validatori basata su criteri puramente legati alla quantità di moneta che gli utenti posseggono, questa è resa efficiente ma meno sicura rispetto all'approccio PoW. Infatti questo meccanismo di elezione rende il risultato del processo in parte prevedibile, esponendo i nodi eletti a potenziali attacchi. Inoltre, un sistema di questo tipo tende ad eleggere una commissione ristretta di validatori, andando a compromettere anche la decentralizzazione della blockchain.

Dagli esempi precedenti emergono tre proprietà fondamentali che una blockchain deve possedere: scalabilità, sicurezza e decentralizzazione. Garantire contemporaneamente tutte e tre le proprietà risulta ad oggi impossibile, ed è un fenomeno noto come *Blockchain Trilemma*. Una recente proposta da parte del premio Turing e noto crittografo Silvio Micali, punta a risolvere il suddetto trilemma con la blockchain *Algorand*, la quale come vedremo utilizza un algoritmo di consenso innovativo che è allo stesso tempo sicuro, scalabile, e garantisce una decentralizzazione effettiva del sistema.

1.1 Scopi e motivazioni

Lo scopo di questo tirocinio è quello di studiare la blockchain di Algorand, con particolare attenzione all'algoritmo di consenso che utilizza, chiamato *Pure Proof of Stake*. Alla base di questo innovativo algoritmo vi è l'utilizzo di una tecnica crittografica molto sofisticata, che permette di eseguire il processo di elezione e validazione con estrema efficienza e sfruttando il concetto di stake in un modo che riduce al minimo le probabilità di prevederne il risulta-

¹<https://eos.io/>

²<https://cardano.org/>

³<https://ethereum.org/en/eth2/>

to. La funzione crittografica in oggetto è chiamata *Verifiable Random Function*[3] (VRF) ed è utilizzata per implementare una sorta di lotteria che elegge la commissione validatrice in modo casuale. Nel seguito del documento vedremo come questo approccio renda la blockchain Algorand estremamente rapida nella validazione delle transazioni, mantenendo un alto livello di decentralizzazione e di sicurezza. Sono state poi confrontate le caratteristiche dell'algoritmo PPoS con le principali alternative già annunciate, come PoW, DPoS e BPOS. Una volta comprese le potenzialità dell'algoritmo PPoS, si è passati ad analizzare l'architettura della rete Algorand e le features di cui dispone, con particolare focus agli Smart Contract ed altre primitive disponibili al layer-1, ovvero funzionalità eseguite on-chain sotto forma di transazioni. Durante questa fase del lavoro è stato capito come la combinazione delle funzionalità di base possa implementare un caso d'uso generale complesso. Tuttavia sono emerse anche delle limitazioni di questa tecnologia, probabilmente dovute al fatto che essa si trova in uno stato iniziale dello sviluppo e non dispone per tanto degli strumenti sofisticati che possiamo invece trovare su blockchain più evolute, come ad esempio Ethereum.

In una fase successiva del lavoro si è passati allo studio delle possibili applicazioni di questa blockchain, prendendo come punto di riferimento e di confronto la blockchain Ethereum, la quale ad oggi rappresenta la principale piattaforma per lo sviluppo di applicazioni decentralizzate e strumenti finanziari. Infatti, oltre a capire quali siano le innovazioni che Algorand porta con sé, era necessario capire anche come queste innovazioni potessero essere impiegate in uno scenario pratico e reale. Per fare ciò è stato sviluppato un porting di una nota applicazione già presente su Ethereum, di modo da effettuare un confronto a 360 gradi con la nuova blockchain. In questa fase ci si è concentrati prima di tutto alla comprensione dei paradigmi utilizzati nell'implementazione dell'applicazione originale su Ethereum. Successivamente si è passati alla progettazione di un'architettura che permettesse di riportare su Algorand le principali funzionalità dell'applicazione in oggetto, mantenendone la semantica.

Per riuscire a fare ciò è stato necessario ripensare integralmente il modo in cui l'applicazione implementava le varie funzionalità. Infatti Algorand, al contrario di Ethereum, non dispone di Smart Contract paragonabili ad una comune applicazione che segue il paradigma a oggetti, bensì adotta un linguaggio non Turing-completo, ad interpretazione sequenziale, che non offre costrutti iterativi o strutture dati complesse. Questa caratteristica limitante di

Algorand ha richiesto l'uso di approcci alternativi che conducessero ai risultati desiderati, permettendo così di sfruttare a pieno le principali caratteristiche della blockchain. Infine le due implementazioni sono state messe a confronto analizzandone vari aspetti, e mettendo in risalto vantaggi e svantaggi di entrambe.

1.2 Struttura del documento

Questo documento è organizzato come segue: il Capitolo 2 presenta lo stato dell'arte delle tecnologie blockchain, fornendo prima al lettore le nozioni base di crittografia, e passando poi ad una breve descrizione degli elementi tipici di una blockchain. Nel Capitolo 3 si descriverà in modo dettagliato la blockchain Algorand, partendo dal concetto di *Blockchain Trilemma* e descrivendo gli approcci utilizzati su Algorand per porvi rimedio; a seguire una descrizione delle features e degli strumenti di sviluppo disponibili su questa piattaforma. Il Capitolo 4 presenterà prima la nota applicazione *CryptoKitties* sviluppata su Ethereum, e descriverà il porting della stessa su blockchain Algorand. A seguire il Capitolo 5 mostrerà i risultati ottenuti dal confronto tra le due implementazioni al fine di poter valutare attraverso dati concreti quali siano gli effettivi vantaggi della nuova tecnologia rispetto alla precedente. Infine nel Capitolo ?? si discuteranno i risultati ottenuti e si valuteranno alcuni scenari alternativi di utilizzo della blockchain Algorand.

Capitolo 2

Stato dell'arte

In questo capitolo il lettore verrà introdotto ai concetti fondamentali necessari a comprendere a pieno gli argomenti trattati nel seguito. Il capitolo inizia con una breve descrizione delle nozioni crittografiche (Sezione 2.1) utilizzate nello sviluppo di tecnologie blockchain e continua poi con la definizione di blockchain vera e propria (Sezione 2.2), fornendo le caratteristiche che la contraddistinguono (Sezioni 2.2.1 e 2.2.2). Successivamente faremo una panoramica di come questa tecnologia si sia evoluta negli anni (Sezione 2.3) e cosa questo comporti per il mondo informatico. Tratteremo poi alcune delle problematiche della tecnologia delle blockchain (Sezione 2.2.3). Infine descriveremo il concetto di asset (2.4) utilizzato nelle DApps e ripreso nei capitoli successivi.

2.1 Fondamenti di crittografia

Definizione. (P2P) Una rete peer-to-peer (P2P) è una rete formata da nodi collegati tra di loro in modo non gerarchico, ovvero ognuno di essi ha un ruolo paritario all'interno della rete. Un esempio noto di reti P2P è quello dei servizi di file sharing (e.g. BitTorrent), nei quali ciascun utente (detto "peer" dall'inglese "pari") funge al contempo sia da cliente che da server, ricevendo e condividendo contemporaneamente i dati con il resto della rete.

Definizione. (Funzione di Hash) Una funzione di hash è una particolare funzione matematica che genera, a partire da un input formato da dati arbitrari, un risultato univoco chiamato "digest". Il digest prodotto con il processo di "hashing" è una stringa binaria di dimensione fissa (indipendente dalla dimensione del dato in input) e non invertibile, ovvero non è possibile rincondursi al dato di origine a partire dal valore dell'hash calcolato. Si tratta quin-

di di una funzione matematica con forti proprietà di sicurezza che viene spesso utilizzata in ambito crittografico. Il digest è facile da calcolare, riproducibile e irreversibile, il che rende questa funzione ideale quando si ha la necessità di garantire la consistenza dei dati. Tra la più diffusa funzione di hashing vi è lo SHA-256[4], progettato in modo da ottenere un digest di dimensione fissa pari a 256bit.

Definizione. (Firma digitale) La firma digitale è l'equivalente elettronico della comune firma manuale che apponiamo sui documenti, al fine di garantire la nostra sottoscrizione ai termini riportati sul documento. Allo stesso modo la firma digitale è uno strumento crittografico volto a garantire delle proprietà di sicurezza nello scambio di messaggi:

- *Autenticità*: il mittente del messaggio è chi dice di essere;
- *Non ripudio*: una volta apposta la firma sul dato, il mittente non può negare di esserne effettivamente l'autore;
- *Integrità*: il messaggio giunto al destinatario non è stato modificato lungo il percorso.

La firma digitale si basa su delle proprietà matematiche che stanno alla base della cosiddetta "crittografia asimmetrica", anche detta crittografia a chiave pubblica-privata, il cui funzionamento è descritto brevemente nel seguente schema:

$$\text{sign}(M, \text{private_key}) = S$$

$$\text{verify}(M, S, \text{public_key}) = \text{True/False}$$

Il mittente genera la firma S relativa al messaggio M utilizzando la propria chiave privata, e spedisce al destinatario la coppia $\langle M, S \rangle$. A questo punto il destinatario utilizza la chiave pubblica del mittente per decifrare S e verificare che il risultato corrisponda effettivamente al messaggio originario M .

Definizione. (Zero-Knowledge) Il protocollo zero-knowledge è un metodo crittografico volto a permettere ad un "prover" P di dimostrare ad un "verifier" V che si trova in possesso di una determinata facoltà o conoscenza, senza condividere nessun'altra informazione in merito [4].

Spieghiamo il concetto alla base di questo protocollo con un semplice esempio:

- Supponiamo che P voglia dimostrare a V che è in grado di contare il numero di granelli di sabbia presenti in una qualsiasi spiaggia (per semplicità assumiamo che

sappia solamente dire se il numero di granelli è pari o dispari), senza però rivelare quale metodo utilizza per il conteggio dei granelli;

- V sceglie un numero di iterazioni k nelle quali P dovrà rispondere a dei quesiti posti da V ;
- A questo punto V lancia una moneta, e a seconda del risultato e del lancio (quindi con probabilità $p = \frac{1}{2}$) decide di rimuovere o meno un granello

$$e = \begin{cases} 1, \text{ rimuove il granello} \\ 0, \text{ non rimuove il granello} \end{cases}$$

lontano dagli occhi di P che sarà quindi all'oscuro della scelta di V ;

- Poi P riesegue la conta dei granelli per quella specifica iterazione e comunica il valore calcolato b_i a V ;
- Sia b_i il valore (pari/dispari) calcolato da P all'iterazione i -esima e b_{i-1} quello relativo all'iterazione precedente, con $i = 1, \dots, k$, allora V esegue la verifica con il criterio:

$$\begin{cases} (e = 1 \ \& \ b_i \neq b_{i-1}) \text{ oppure } (e = 0 \ \& \ b_i = b_{i-1}) \implies \text{ iterazione successiva} \\ \text{altrimenti interrompi il processo (verifica fallita)} \end{cases}$$

Definizione. (Verifiable Random Function)[3] Una Verifiable Random Function (VRF) è una funzione crittografica che prende un dato in input x e produce:

- Un output pseudo-casuale associato all'input x e ad un valore segreto che detiene solo l'esecutore;
- Un valore detto $VRF - proof$ che attesta la correttezza di tale computazione senza esporre la chiave segreta utilizzata nel calcolo.

Queste funzioni permettono il controllo asincrono del risultato di una computazione con un paradigma simile allo *zero-knowledge*. Più in dettaglio una VRF si compone di 3 parti (funzioni):

- **Keygen:** dato un input r l'algoritmo produce una *verification key* V_K ed una *secret key* S_K

$$(V_K, S_K) \leftarrow \text{Keygen}(r)$$

- **Evaluate:** prende in input S_K ed un valore x e produce un valore pseudo-casuale y ed una *proof* p

$$(y, p) \leftarrow \text{Evaluate}(S_K, x)$$

- **Verify:** prende in input V_K , x , y e p e restituisce:

$$\begin{cases} 1, \text{ se } y \text{ è il risultato corretto di } \text{Evaluate}(S_K, x) \\ 0, \text{ altrimenti} \end{cases}$$

2.2 Blockchain

Una blockchain è un registro digitale *distribuito* e *immutabile*, talvolta riferito in letteratura con il nome di *Distributed Ledger*. Come suggerisce il nome stesso si tratta di un registro (ledger) formato da blocchi concatenati, ciascuno dei quali contiene un'insieme di transazioni effettuate nel corso del tempo da tutti i nodi facenti parte della rete P2P. Questa tecnologia nasce alla fine del 2008 da un'idea rivoluzionaria dell'inventore noto con lo pseudonimo di Satoshi Nakamoto[1], il quale aveva come obiettivo quello di creare un ecosistema finanziario privo di istituti di controllo. Nacque così l'idea di creare un sistema nel quale gli utenti potessero effettuare pagamenti elettronici, senza dover passare da un ente centrale che garantisse la correttezza delle parti in cambio di un compenso per la mediazione del pagamento.

L'utilizzo di un registro distribuito, infatti, permette a tutti gli utenti della rete di registrare le proprie transazioni su di esso, nonché consultare i dati già registrati in precedenza anche da altri utenti. In questo modo vi è una consapevolezza condivisa (ciascun nodo della rete possiede una copia del registro) che non può essere ripudiata da nessuno dei membri. Il problema da affrontare in un contesto di questo tipo, basato sulla gestione decentralizzata delle informazioni, è quello di porre dei limiti e delle regole comuni volte a garantire la sicurezza di tutti i partecipanti. Per garantire queste proprietà in un ambiente decentralizzato, e quindi privo di una entità centrale delegata al controllo, sono stati sviluppati gli *Algoritmi di consenso*.

L'algoritmo di consenso è quello che definisce chi, come e cosa viene scritto sul ledger, ed è pertanto il fulcro intorno al quale viene costruita l'architettura di una blockchain. Con la prima blockchain (Bitcoin, 2008), Nakamoto introdusse un protocollo volto a garantire (1) la legittimità dei dati contenuti nella transazione, (2) l'immutabilità dei dati nel tempo.

Per comprendere il funzionamento di questo protocollo si consideri la rappresentazione della blockchain in figura 2.1:

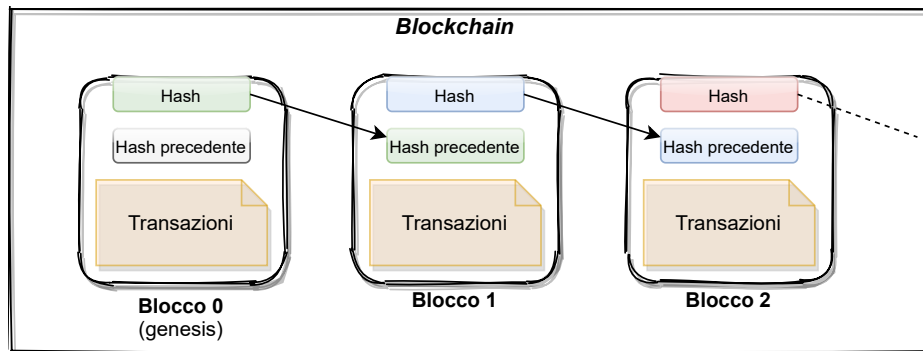


Figura 2.1: Schema semplificato di una blockchain

Ciascun blocco contiene un insieme di transazioni, un digest del blocco precedente (0 nel caso del genesis block) ed un digest del blocco corrente, quest'ultimo calcolato utilizzando come input l'insieme delle transazioni ed il valore del digest precedente. Questo meccanismo oltre a creare un collegamento logico tra i vari blocchi, rappresenta ciò che garantisce l'immutabilità dei dati. Inoltre su ciascuna transazione vi è la firma digitale associata all'utente che ha sottoposto quella transazione, la quale rappresenta un'identificativo univoco per ciascun utente e per ciascuna transazione di un dato utente, garantendo così tutte le proprietà viste in Sezione 2.1 e quindi la singola transazione. Se anche una sola transazione fosse manomessa, il valore hash del blocco di appartenenza risulterebbe invalido, ed a catena anche tutti i blocchi successivi.

Serve poi un protocollo volto a validare questi dati prima che possano essere riportati in modo permanente sulla blockchain. Ciascun nodo della rete raccoglie un insieme di transazioni e le raggruppa in un blocco che intende scrivere in modo permanente sulla blockchain. L'algoritmo di consenso decide poi chi tra tutti i nodi proponenti può acquisire il diritto di memorizzare il proprio blocco sul ledger: tale algoritmo deve garantire l'equa probabilità a tutti i partecipanti di essere scelti. Il primo algoritmo di consenso sviluppato in questo ambito¹ fu proprio quello di *Bitcoin* proposto da Nakamoto stesso: l'algoritmo prevede che il nodo che vuole inserire un blocco sul ledger, debba risolvere un puzzle crittografico di difficoltà crescente nel tempo. Questo approccio, che prende il nome di **Proof of Work (PoW)**[5], prevede che il nodo che propone il nuovo blocco (detto *miner*) riesca a trovare

¹approcci simili furono già utilizzati a partire dagli anni '90 per scopi differenti: <https://en.wikipedia.org/wiki/Hashcash>

un valore (detto *nonce*) che posto in hash con il gruppo di transazioni all'interno del blocco, restituisca un digest le cui t cifre iniziali siano pari a 0 (con t parametro che varia al crescere della potenza di calcolo dei miner). Se il blocco proposto, ed il valore del *nonce* calcolato dal *miner* proponente, riceve l'approvazione dalla maggioranza dei nodi della rete, il valore del digest sarà inserito nel campo "hash" del blocco, che infine verrà scritto permanentemente sulla blockchain andando a formare la nuova versione della blockchain che sarà poi aggiornata localmente dalla totalità dei nodi.

Tuttavia il sistema presenta degli svantaggi non indifferenti, tra cui il costo computazionale (e di conseguenza energetico) richiesto per il calcolo del *nonce*. Questa caratteristica infatti ha portato alla formazione di così detti *Mining Pools*, ovvero agglomerati di nodi che collaborano per raggiungere la potenza di calcolo necessaria a risolvere i suddetti puzzle crittografici ed assicurarsi la possibilità di scrivere sulla blockchain [6]. Come si evince

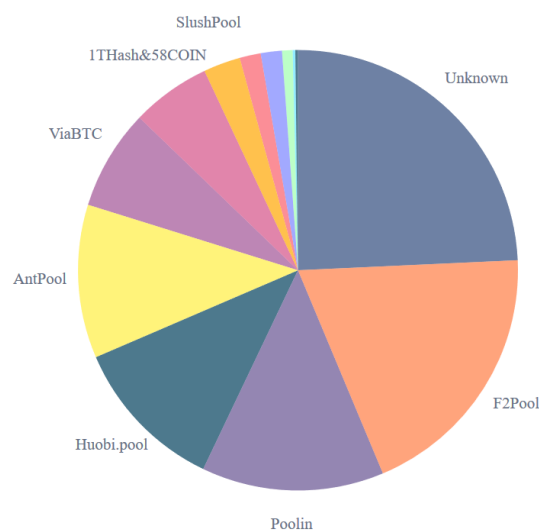


Figura 2.2: Principali mining pools della rete Bitcoin

dalla Figura 2.2 nel corso degli anni si è creata una forte concentrazione di miners che hanno acquisito il controllo della rete, facendo riemergere esattamente lo stesso problema che Nakamoto stava cercando di risolvere, ovvero la centralizzazione. È evidente infatti che una rete nella quale una grossa fetta di miners, tutti sotto un unico dominio/organizzazione, possiedono la potenza di calcolo necessaria a controllare di fatto l'intera rete, non può di certo definirsi decentralizzata. Infatti in teoria tutti i nodi possono partecipare alla corsa per la risoluzione del puzzle, ma nella pratica solamente pochi dispongono delle risorse necessarie a svolgere tale compito.

Da qui nacque una nuova famiglia di algoritmi di consenso che va sotto il nome di **Proof**

of Stake (PoS)[7]. Come suggerisce il termine stesso, non si tratta più di un approccio che prevede l'esecuzione di grossi calcoli (da cui Proof of Work), bensì si tratta di mettere in gioco i propri possedimenti (*stakes*). Descriviamo brevemente le differenze tra queste due famiglie di algoritmi in quanto queste nozioni saranno fondamentali per comprendere quanto trattato nelle prossime sezioni.

2.2.1 PoW vs PoS

Come anticipato, la Proof-of-Work presenta delle limitazioni in termini di scalabilità, nonché la tendenza ad introdurre una forma di centralizzazione nel sistema, ed è stata pertanto sostituita dagli approcci Proof-of-Stake a partire dal 2011. Esistono vari tipi di algoritmi di consenso basati sul PoS, ma in generale si basano tutti quanti sulla selezione di chi proporrà e validerà il blocco successivo. Tale nodo prende in questo contesto il nome di *minter* o *forger*, mentre il processo di validazione viene detto *forging*.

La scelta del candidato avviene per lo più sulla base della posta che ha messo in gioco, infatti, maggiore è lo stake del nodo (quantità di coin congelate per il processo di validazione) maggiore è la perdita che quel nodo avrebbe qualora agisse in modo scorretto, per cui in modo inversamente proporzionale garantisce la maggiore affidabilità di quel *minter* rispetto agli altri. Anche in questo caso il protocollo prevede che a seguito della scelta del nodo proponente vi sia il consenso da parte di un numero arbitrario di nodi che agiscono come una sorta di commissione (nel caso del PoS standard è l'intera rete ad agire da commissione e votare quindi a favore o contro la scrittura del blocco). Il criterio con il quale viene scelto il nodo validatore e la commissione è diverso per ciascuna variante dell'algoritmo PoS. Vedremo questi algoritmi e le loro caratteristiche più in dettaglio nel seguito di questo documento.

2.2.2 Delegated vs Bonded PoS

Le due principali evoluzioni del metodo PoS sono la *Delegated Proof of Stake* [8] e la *Bonded Proof of Stake* [9], brevemente descritti di seguito per dare al lettore una chiave di lettura per quanto trattato nella sezione successiva.

- **Delegated PoS:** a differenza del metodo PoS, in cui la convalida delle transazioni prevede il coinvolgimento dell'intera rete, il DPoS delega tale compito ad una cerchia

ristretta di nodi che prende il nome di *commissione*. Il criterio utilizzato per l'elezione della commissione è simile a quello utilizzato in ambito politico con la democrazia rappresentativa. L'intera rete vota i delegati in base allo stake che possiedono, ed il voto inoltre assume un peso differente anch'esso sulla base dello stake posseduto dal votante. Come è stato detto in precedenza, infatti, questi sistemi basano la loro sicurezza proprio sulla quantità di token appartenenti al candidato come forma di garanzia del loro operato [DPoS].

- **Bonded PoS:** in questa variante del PoS non vi è una commissione di *delegati* eletti dalla rete come per la variante DPoS, bensì ciascun nodo si "candida" depositando parte dei suoi token, congelandoli per un lasso di tempo predefinito. Allo scadere del tempo di deposito, ciascun nodo ha una probabilità di essere scelto come minter pari alla quantità di token depositati (il suo stake appunto). Questo deposito rappresenta ancora una volta una forma di garanzia per la rete, in quanto rappresenta la perdita monetaria che il nodo avrebbe qualora agisse in modo scorretto, e quindi tanto più grande è lo stake → maggiore il rischio per il minter → maggiore la garanzia per la rete.

Esistono altre varianti del PoS meno diffuse, riportate per completezza nella seguente figura:


Overview of PoS Variants							 Stakin
	DPoS	NPoS	LPoS	BPoS	DPoC	HPoS	Others
Voting Rights	Vote for Validators ONLY	Vote for protocol amendments	Vote for protocol amendments	Vote for protocol amendments	Vote for Validators ONLY	Depends on the protocol	Other alternatives include Masternode, staking, Leased PoS, PoS Voting, Delegated BFT.
Nb. Validators	Limited in quantity & elected	100+, limited in quantity by governance	Min. Bond Size	Limited in quantity & selected by stake size	Paid by Validator & sometimes Delegator	Min. Bond Size	
Security Sanctions	Depends on the protocol	Paid by Validator & Delegator	Paid by Validator	Paid by Validator & Delegator	Paid by Validator	Depends on the protocol	Depends on the protocol
Examples	Lisk Eos Steem Bitshares	Polkadot Kusama Edgeware	Tezos	Cosmos Network IRISNet	ICON	Decred Hcash Ethereum Casper	Dash Waves NEO Livepeer

Figura 2.3: Panoramica delle varianti PoS [10]

2.2.3 Blockchain forks

Quando si parla di blockchain, soprattutto quelle basate sulla Proof of Work come Bitcoin ed Ethereum, un fattore da tenere a mente è quello delle *blockchain fork* o *forking*. Vediamo ora in cosa consiste questo fenomeno, perché si verifica, e perché rappresenta un problema per gli utenti. Proviamo ad immaginare cosa accade durante il processo di *mining* dei blocchi su Bitcoin (varrà lo stesso per Ethereum):

- Molteplici nodi si mettono a lavoro per la risoluzione del puzzle crittografico ed aggiudicarsi la scelta del successivo blocco da validare;
- Eventualmente un nodo troverà la soluzione, sceglierà il blocco da minare e lo propagherà alla rete. Se invece dovesse accadere che due o più nodi trovino la soluzione contemporaneamente, entrambi propagheranno i rispettivi blocchi proposti alla rete, ignari di essere contendenti;
- Chiamiamo i due vincitori N_1 ed N_2 ed i rispettivi blocchi proposti B_1 e B_2 . Sia N_1 che N_2 procederanno normalmente, propagando i loro rispettivi blocchi alla rete. Di fatto loro non sono a conoscenza del fatto di essere contendenti, per cui agiscono come se si trovassero nel caso 1.
- I due blocchi B_1 e B_2 arrivano agli altri nodi della rete in ordine casuale, prima B_1 e poi B_2 o viceversa per ciascun nodo. Chi riceverà per primo il blocco B_1 considererà questo come blocco da aggiungere alla propria copia del ledger, e analogamente per chi riceverà per primo il blocco B_2 . Di conseguenza la blockchain avrà due stati differenti allo stesso tempo come mostrato nella figura seguente:

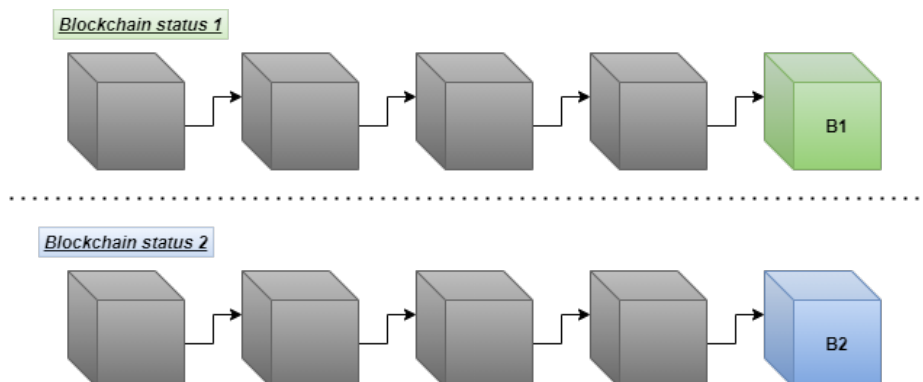


Figura 2.4: Formazione di stati coesistenti

La coesistenza di due stati differenti della blockchain è quello che dà origine alla biforcazione, innescando il seguente processo:

- Ciascun nodo della rete si ritrova a questo punto a dover scegliere quale dei due percorsi considerare valido, e quale invece da scartare. La scelta su Bitcoin è in un certo senso democratica, nel senso che se il 51%+ dei nodi sceglie di proseguire da B_1 allora i blocchi successivamente minati da quella parte di rete verranno aggiunti alla catena 1, viceversa la controparte che ha scelto di proseguire da B_2 sceglierà quest'ultimo come blocco da cui ripartire. Inevitabilmente la maggior parte dei nuovi blocchi saranno aggiunti alla prima catena, in quanto è maggiore la possibilità di risolvere il puzzle da parte della fetta più grande della rete;
- Una volta raggiunta una differenza di lunghezza predefinita, tutti i nodi si addegueranno nell'adottare il percorso che ha riscosso più successo (quello al quale sono stati aggiunti più blocchi) e a scartare quello più corto, invalidando tutto il lavoro compiuto a partire dallo split fino a quel punto.

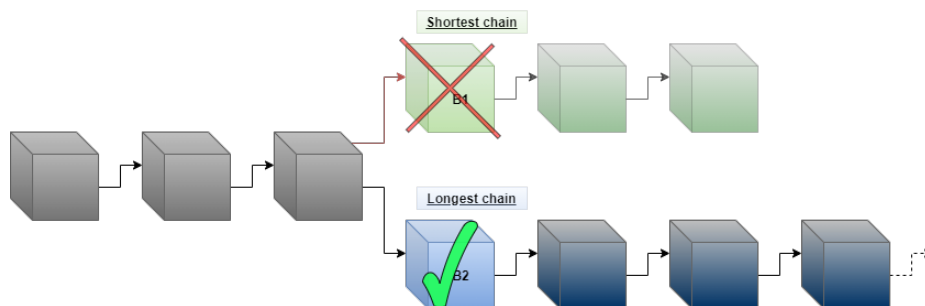


Figura 2.5: Scelta del percorso più lungo

Questo fenomeno rappresenta un problema sotto diversi aspetti:

- Come già detto, effettuare il mining dei blocchi è un processo molto dispendioso in termini di tempo e di costi, costi che vanno ripagati (con un certo margine di guadagno) tramite ricompense in cambio del lavoro svolto per validare le transazioni del blocco. Queste ricompense verranno annullate insieme alle transazioni da cui derivavano, e risulta chiaro dunque che scartare un intero ramo della catena rappresenta una perdita enorme per tutti coloro che hanno contribuito alla sua creazione spendendo le proprie risorse;

- Altro fattore non trascurabile è l'effetto che questo fenomeno ha dal punto di vista di chi usufruisce dei servizi sviluppati sulla blockchain. Ricordiamo che i blocchi sono solamente l'involucro dei dati che contengono (le transazioni). Scartare uno dei rami della catena significa buttare via i blocchi con tutto ciò che contengono, e quindi comporta l'invalidazione di tutte le transazioni ivi contenute. Immaginiamo ad esempio una transazione di pagamento effettuata in favore di un fornitore: la transazione era stata approvata e riportata sul ledger, ma sfortunatamente sul ramo sbagliato, e quindi è come se quel pagamento non fosse mai avvenuto. Per evitare situazioni sgradevoli di questo tipo vengono attuate delle politiche cautelative che prevedono un certo tempo di attesa prima di poter considerare una transazione effettivamente approvata, pur trovandosi già scritta sul ledger;
- La conseguenza che scaturisce da queste problematiche è di voler limitare quanto più possibile questi fenomeni abbassando la probabilità che il puzzle possa essere risolto troppo frequentemente o con troppa facilità. Alzando il livello di difficoltà di mining però, si peggiora la situazione in termini di lentezza della rete, potenza di calcolo necessaria e consumo energetico.

2.3 Evoluzione delle Blockchain

A seguito della nascita delle blockchain, si è pensato non solo a come migliorarne la scalabilità, sicurezza e decentralizzazione, ma anche ad accrescerne le funzionalità di modo da poter impiegare questa tecnologia in ambiti diversi da quello finanziario. Con l'avvento della seconda blockchain più popolare al mondo, cioè *Ethereum* [2], è stato introdotto un sistema che permette di programmare le blockchain utilizzandone la tecnologia sottostante per poter rappresentare non solo una valuta finanziaria, ma anche oggetti più complessi come ad esempio delle *custom currencies*, la proprietà fisica di beni o la proprietà intellettuale di beni astratti (come ad esempio il copyright su produzioni artistiche) ecc. Sono nati così gli *Smart Contract*² su rete Ethereum: dei "contratti intelligenti" scritti in un linguaggio Turing completo (*Solidity* nel caso specifico di Ethereum) che implementano la logica con la quale vengono rappresentati e gestiti questi oggetti direttamente sulla blockchain,

²l'idea di smart contract fu esposta per la prima volta nel '97 da N. Szabo: <https://firstmonday.org/ojs/index.php/fm/article/view/548>

andando a sfruttare le proprietà di decentralizzazione e sicurezza di questa tecnologia.

Gli smart contracts su Ethereum sono delle vere e proprie applicazioni eseguite su una macchina virtuale che prende il nome di *Ethereum Virtual Machine (EVM)*. La EVM è implementata all'interno di ogni nodo della rete in aggiunta al software che gestisce il protocollo di base, il quale si occupa della gestione delle transazioni e dell'algoritmo di consenso, ed è adibita all'esecuzione di queste applicazioni.

Con l'introduzione di questi strumenti è stato reso possibile creare delle applicazioni software decentralizzate, le *DApps (Decentralized Applications)*, che vengano eseguite direttamente sulla rete e risultino quindi sicure e trasparenti a tutti gli utilizzatori del servizio che forniscono. Si pensi ad esempio alla gestione dei diritti d'autore su opere multimediali³, o ancora ai sistemi di raccomandazione e recensione di beni e servizi, oppure al tracciamento di beni materiali, o ancora alla registrazione delle varie fasi del processo produttivo relativo ad un prodotto. Tutti quanti sistemi ad oggi gestiti e controllati da un provider centrale del servizio che possono invece essere portati su blockchain rimuovendo i problemi di sicurezza intrinseci dei sistemi centralizzati.

Il concetto di "transazione" diventa quindi più ampio, e non si limita più alla semplice rappresentazione di uno scambio finanziario bensì ad oggetti più complessi contenenti dati arbitrari, validati ed immutabili nel tempo. Ethereum rappresenta un punto di riferimento per le tecnologie blockchain proprio per le enormi potenzialità dei suoi Smart Contracts, attraverso i quali sono state sviluppate DApps di varia natura che hanno apportato grandi benefici alle attività che ne usufruiscono, dalla gestione di una catena di montaggio al monitoring e tracking di merci, e molte altre ancora⁴. Tuttavia anche Ethereum come Bitcoin utilizza un protocollo basato su PoW (almeno nella versione attuale) trascinando con sé quindi tutte le limitazioni e debolezze del caso. Diventa sempre più chiaro come non sia affatto semplice soddisfare tutti gli aspetti presenti in un sistema apparentemente semplice, ma che in realtà costituisce una continua corsa al perfezionamento. Nel capitolo successivo esamineremo uno dei più audaci tentativi di creare una blockchain prossima a risolvere tutte le problematiche descritte sino ad ora: la blockchain Algorand.

³"L'arte certificata: <https://www.ilpost.it/2021/03/03/arte-blockchain-nft/>

⁴Ethereum Dapps: <https://www.coindesk.com/7-cool-decentralized-apps-built-ethereum>

2.4 Fungible & Non-Fungible tokens

Con l'evolversi delle *Distributed Ledger Technologies (DLT)*, ossia delle applicazioni decentralizzate basate su blockchain, si è reso necessario poter rappresentare oggetti astratti come il copyright, game collectibles, game coins, domini, valute proprietarie e contenuti digitali in generale. Per assolvere a queste necessità Ethereum ha introdotto il concetto di *asset*, ovvero dei token non nativi (diversi dagli *ETH*) personalizzabili e generabili ad-hoc per la specifica applicazione [11]. Questi asset si dividono in due categorie, di seguito descritte:

- **Fungible Tokens**

I fungible tokens sono molto simili alla cryptovaluta nativa della blockchain (che in alcuni contesti viene trattata esattamente come un fungible token) ed è semplicemente una moneta virtuale con un corrispettivo valore in ETH utilizzata all'interno dell'applicazione come valuta proprietaria. Si pensi ad esempio al mondo videoludico che sempre più spesso inserisce una valuta fittizia all'interno dell'ecosistema, utilizzata dagli utenti per l'acquisto di prodotti in-game o sullo store del produttore. Il valore di ciascun token, la quantità di tokens conati e l'utilizzo che è possibile farne sono tutti parametri personalizzabili, a differenza degli ETH che dipendono dall'andamento del mercato ed altri fattori non manipolabili. Un fungible token può essere scambiato con un altro fungible token dello stesso tipo e sono quindi entità interscambiabili, cioè ogni coin ha lo stesso valore, proprio come avviene per le valute fisiche come il dollaro o l'euro che tutti conosciamo.

- **Non-Fungible Tokens**

Ben più interessanti sono invece i *Non-Fungible tokens (NFT)*, utilizzati per rappresentare oggetti unici e non interscambiabili. Ciascuno di questi token è rappresentativo di una singola entità fisica o astratta (un game item, un diritto d'autore su di un'opera o creazione, la proprietà di un bene fisico ecc.). Come per i normali token della categoria precedente, anche i NFT permettono la personalizzazione dei parametri (quale oggetto rappresentano, che valore ha l'oggetto, chi è il proprietario dell'oggetto ecc.) e sono per tanto degli strumenti molto utilizzati in varie applicazioni. I NFT sono definiti e manipolati su Ethereum tramite dei particolari Smart Contract che fungono da interfaccia e fissano quali caratteristiche deve avere un asset per esser definito

un NFT. Lo standard attuale è lo ERC-721⁵, ma ve ne sono di altri utilizzati in ambienti specifici, come ad esempio lo ERC-1155⁶ utilizzato per applicativi videoludici e lo EIP-2309⁷ utilizzato per facilitare il processo di creazione dei token.

⁵ERC-721: <http://erc721.org/>

⁶ERC-1155: <https://eips.ethereum.org/EIPS/eip-1155>

⁷EIP-2309: <https://eips.ethereum.org/EIPS/eip-2309>

Capitolo 3

Algorand

Nel capitolo precedente abbiamo visto come la tecnologia blockchain abbia introdotto nuovi paradigmi nel mondo informatico ed abbia permesso di risolvere le principali problematiche legate alla gestione centralizzata di un determinato servizio.

Tuttavia non si è ancora giunti ad uno stato in cui questi sistemi possano essere utilizzati su vasta scala in applicazioni complesse. Ciascuna blockchain, infatti, con il proprio algoritmo di consenso presenta dei punti deboli derivanti dalla necessità di dover scegliere se puntare sulla scalabilità piuttosto che sulla sicurezza, sulla decentralizzazione piuttosto che sulla scalabilità e così via. Questo problema è conosciuto in ambito blockchain come *Blockchain Trilemma* [12], ovvero l'impossibilità di garantire contemporaneamente tutte e tre le proprietà. In questo capitolo vedremo prima in cosa consiste il trilemma e la soluzione proposta da Algorand (Sezione 3.1), per poi passare all'infrastruttura della rete P2P di Algorand (Sezione 3.2). Successivamente verranno descritte le features disponibili ad oggi sulla blockchain, tra cui gli Smart Contracts (Sezione 3.3) e gli asset (Sezione 3.4), ed infine esporremo nel dettaglio il concetto di transazione secondo Algorand e gli strumenti disponibili per la loro gestione (Sezione 3.5).

3.1 Blockchain Trilemma

La figura 3.1 mostra come nella costruzione di una blockchain sia necessario scegliere di favorire al più due aspetti tra decentralizzazione, sicurezza e scalabilità, a scapito del terzo. Si pensi ad esempio all'approccio PoW di Bitcoin ed Ethereum, e a come questo abbia favorito da un lato la sicurezza della rete, ma dall'altro ha trascurato sia la scalabilità e a lungo

termine anche la decentralizzazione a causa dell'accentramento del processo di mining. Un altro esempio potrebbe essere quello del DPoS, che risolve sicuramente il problema della scalabilità pur mantenendo un buon livello di decentralizzazione, ma sacrifica la sicurezza a causa della commissione ristretta eletta al fine di validare le transazioni.

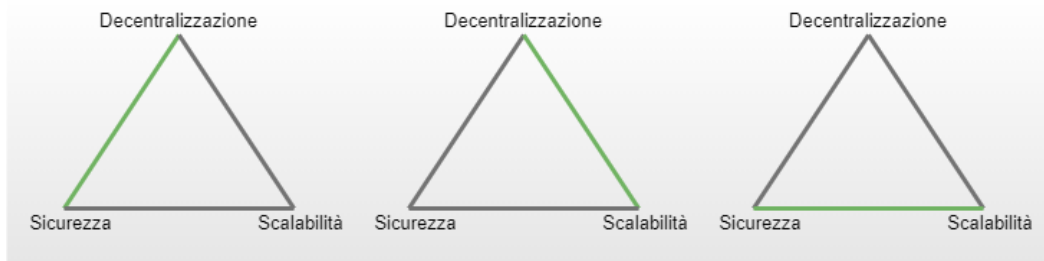


Figura 3.1: Blockchain Trilemma

La soluzione al trilemma sembra esser stata raggiunta da un'idea nata dal premio Turing Silvio Micali agli inizi del 2018, con l'invenzione di una nuova blockchain: Algorand. Come spiegato nel capitolo precedente, il punto cardine di una blockchain è l'algoritmo di consenso che utilizza, ed è proprio su questo aspetto che l'inventore di Algorand ha focalizzato la sua attenzione introducendo una nuova variante del PoS, detto *Pure Proof-of-Stake* (PPoS) [13].

3.1.1 Pure Proof-of-Stake

Il protocollo si basa su una tecnica chiamata *Cryptographic Sortition* implementata tramite una *Verifiable Random Function* (VRF), e si divide in 2 fasi:

1. **Elezione dei nodi;**
2. **Verifica.**

Nella fase di elezione dei nodi vengono selezionati un numero x di token in modo casuale, appartenenti ai nodi della rete. I nodi per i quali è stato selezionato uno o più token partecipano al processo di validazione del blocco, il che significa che più tokens possiede un nodo, maggiore è la sua probabilità di entrare a far parte della commissione. Ciò che distingue questo sistema dagli altri della famiglia PoS è il modo in cui questi nodi (i loro token) vengono selezionati, ed è proprio qui che entra in gioco la VRF:

- Ciascun nodo possiede una "slot machine crittografica" che implementa la VRF, e che non può essere hackerata (crittograficamente sicura);

- Ogni token appartenente al nodo rappresenta un ticket per l'azionamento di tale macchina (ogni token equivale ad una possibilità di vincere la "lotteria" e può essere utilizzato una sola volta durante tutto il processo);
- L'esecuzione della VRF restituisce un risultato (win o loss), ed una prova crittografica che il risultato calcolato sia effettivamente corretto ed associato a quel token ed alla chiave privata del nodo in questione. Tale prova può essere utilizzata da chiunque per verificare in un secondo momento la legittimità del risultato senza dover conoscere la chiave segreta del nodo, e ciò si traduce in una verifica asincrona priva di overhead di sincronizzazione tra i nodi.

Si tratta quindi di un meccanismo di *auto-elezione* dal basso costo computazionale (pochi micro secondi per nodo) e crittograficamente sicuro. Ciascun nodo propaga quindi il proprio risultato con annessa *VRF-proof* che permette ai restanti nodi della rete di verificare la legittimità del risultato ottenuto nella fase di verifica. Vedremo in Sezione 3.1.3 come questo schema di auto-elezione → verifica, viene utilizzato ripetutamente all'interno dell'algoritmo di consenso.

3.1.2 PPOS: Analisi della sicurezza

Abbiamo visto come Algorand abbia risolto il problema della scalabilità e favorito una decentralizzazione effettiva del sistema, grazie al suo algoritmo di consenso innovativo che fa uso di primitive crittografiche ad-hoc. Vediamo come anche il fattore sicurezza sia molto solido ponendoci alcune semplici domande: il protocollo è veramente a prova di attacchi? La delegazione ad un numero ristretto di nodi del processo di validazione, espone la blockchain ai tipici rischi di un sistema nel quale pochi decidono per tutti?

Cominciamo dalla prima domanda, quali sono i possibili attacchi alla sicurezza di una blockchain e come li previene Algorand? Prendiamo in esame il caso del DPoS, algoritmo che prevede l'elezione di una commissione (proprio come per il PPOS): il fatto che la commissione nei sistemi DPoS venga eletta con criteri strettamente legati allo stake, rende in un certo senso prevedibile quali saranno i nodi prescelti. Inoltre l'informazione sul risultato di queste elezioni è resa pubblica a tutti i nodi della rete prima che questi possano selezionare il blocco da validare. Vi è quindi un gap temporale tra la scelta dei nodi e la validazione del blocco. E' proprio in questo frangente che un eventuale attacco può aver luogo, come ad

esempio un attacco *Denial of Service*¹ che taglierebbe fuori dalla rete il nodo attaccato prima che questo possa propagare il suo voto, oppure tramite un semplice tentativo di corruzione. Algorand previene tutto ciò tramite un approccio di elezione "last minute" implementato dall'algoritmo di consenso, trattato in dettaglio nella successiva sezione.

3.1.3 Consensus Algorithm

Segue una descrizione delle varie fasi dell'algoritmo che permetterà al lettore di comprendere meglio i concetti esposti nel paragrafo precedente.

1. Block Proposal

- Viene selezionata una prima commissione con il metodo descritto in Sezione 3.1.1. I vincitori della lotteria propagano al resto della rete sia il blocco che intendono proporre, sia la *VRF-proof* che attesta la loro vittoria. In questo modo non vi è alcun gap temporale tra l'informazione "nodo vincente" e "blocco proposto", e non è possibile dunque condurre un attacco o un tentativo di corruzione nei confronti del nodo in quanto questo ha già propagato la sua scelta nel momento stesso in cui si è annunciato vincitore.

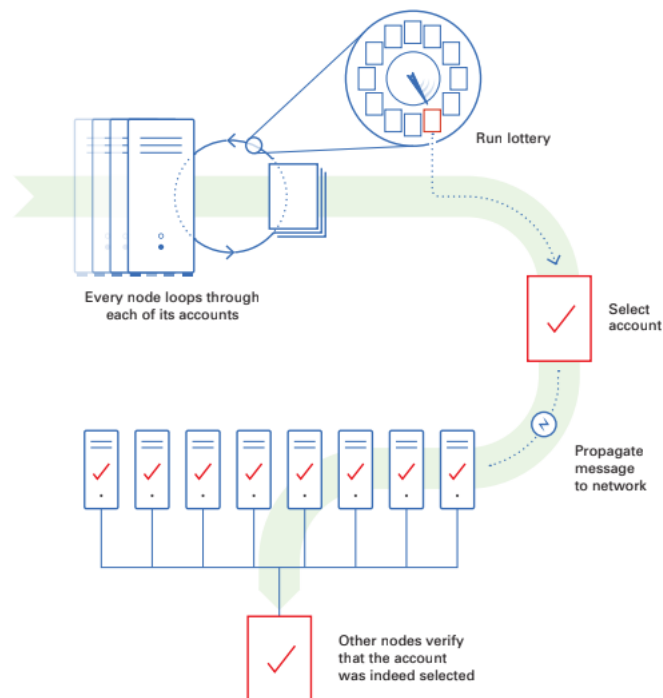


Figura 3.2: Block proposal phase

¹<https://www.cloudflare.com/it-it/learning/ddos/cryptocurrency-ddos-attacks/>

2. Verify (or Soft Vote)

- Viene eseguita una nuova lotteria, questa volta con l'obiettivo di eleggere la commissione che sceglierà uno tra i tanti blocchi proposti nella fase precedente. Ancora una volta i vincitori della lotteria propagano il loro voto in contemporanea alla *VRF-proof* che attesta il loro ruolo. Anche in questo caso dunque il processo di elezione-verifica avviene in modo atomico non lasciando spazio ad eventuali attacchi. Tra tutti i blocchi proposti nella fase precedente, verrà propagato alla rete, e quindi portato in fase di votazione, solamente quello con il valore delle *VRF-proof* più basso, evitando la validazione di blocchi multipli ed eliminando di conseguenza il rischio di creare una biforcazione della blockchain (come avviene ad esempio su Bitcoin e Ethereum).

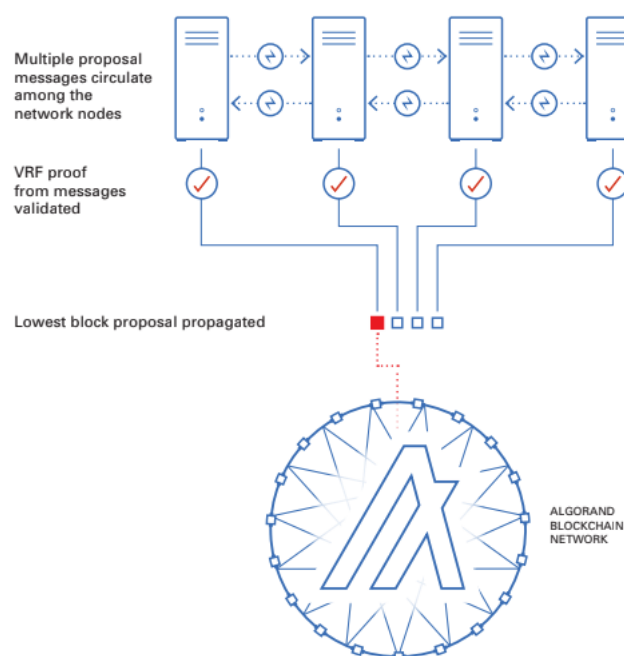


Figura 3.3: Verification phase

3. Certify Vote

- L'algoritmo si conclude con un'ultima esecuzione della lotteria che selezionerà una commissione la quale avrà il compito di verificare la correttezza delle transazioni contenute nel blocco prescelto, e di esprimere il proprio voto a favore o contro la scrittura del blocco. Una volta raggiunto il quorum, il blocco verrà riportato in modo permanente sulla blockchain e propagato alla rete per permettere ai restanti nodi di aggiornare la propria copia del ledger.

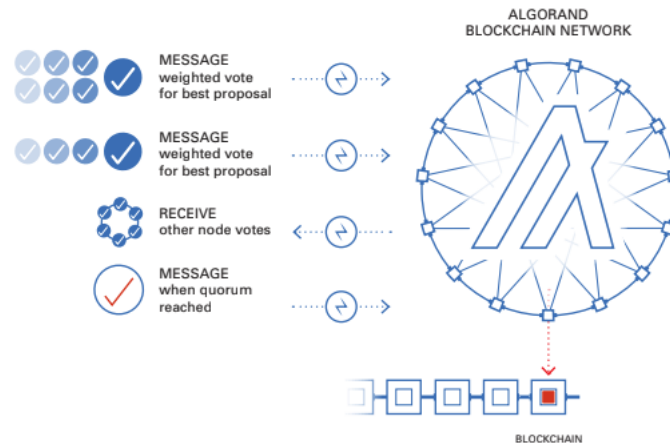


Figura 3.4: Block certification phase

L'algoritmo offre diversi vantaggi rispetto alle controparti PoW, DPoS e BPOS:

- Elezione pseudo-casuale, non prevedibile e democratica (a differenza di DPoS e BPOS in cui l'elezione avviene su base di criteri prefissati rendendo il risultato in parte prevedibile);
- Processo suddiviso in fasi asincrone e atomiche che non lasciano spazio ad eventuali attacchi basati sulla conoscenza a priori dei nodi delegati;
- Funzione crittografica di selezione computazionalmente semplice e al contempo sicura (garantisce elevata scalabilità e robustezza);
- Assenza di *fork* nella blockchain, quindi una volta che il blocco viene riportato sulla blockchain può considerarsi definitivo.

3.2 Architettura di rete

Ora che abbiamo chiarito quali sono gli aspetti teorici su cui si basa il consenso, veniamo alla parte tecnica di questa tecnologia partendo proprio da come è strutturata la rete P2P di Algorand su cui l'intero sistema è costruito. L'infrastruttura si compone di due tipologie di nodi interconnessi, i *Relay nodes* ed i *Participation nodes*, che presentano caratteristiche tecniche e ruoli differenti.

- **Relay nodes**

Hanno il ruolo di hub centrali, ovvero punti di collegamento utilizzati principalmente per il routing dei messaggi da e verso un insieme di non-relay nodes. Questi nodi possono a loro volta essere interconnessi al fine di migliorare le prestazioni della rete in termini di throughput e latenza. Un ulteriore compito svolto da queste entità è il controllo sui messaggi stessi come ad esempio la de-duplicazione delle transazioni, la verifica delle firme digitali apposte sulle transazioni di un blocco ed altre operazioni di verifica e validazione delle stesse. Funendo da nodi portanti per la rete, è necessario che siano ospitati su hardware performante e che dispongano di un collegamento di rete capace di gestire un elevato traffico, requisiti indispensabili per evitare che questi nodi possano rappresentare un "bottleneck" per la rete. Ad oggi diverse compagnie hanno messo a disposizione le proprie risorse per ospitare questi nodi, favorendo una distribuzione uniforme della rete attorno al globo e contribuendo a garantire un alto standard prestazionale [14].



Figura 3.5: Organizzazioni che ospitano la rete Algorand

- **Participation nodes (o non-relay)**

I participation nodes al contrario sono collegati esclusivamente ad uno o più relay nodes ma mai tra di loro. Rappresentano la percentuale maggiore dei nodi che compongono la rete e richiedono poche risorse (salvo casi particolari) in quanto non fungono da punti di routing ma si limitano all'invio di messaggi ed alla ricezione da parte del/degli hub a cui sono collegati.

L'architettura descritta può essere rappresentata con il seguente schema:

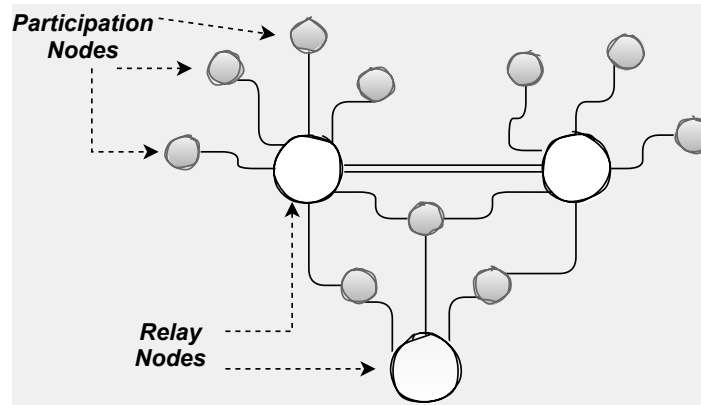


Figura 3.6: Algorand Network Architecture

Entrambe le categorie di nodi presentano opzioni comuni, tra cui quella di poter partecipare al protocollo di consenso (online vs offline mode), quella di mantenere una copia locale del ledger (archival vs non-archival) e di eseguire un servizio demone volto all'indicizzazione delle transazioni presenti sul registro. Maggiori dettagli saranno presentati nel Capitolo 4 dedicato al proof-of-concept sviluppato per lo studio della blockchain.

3.3 Algorand Smart Contracts (ASC1)

Nella Sezione 2.3 abbiamo introdotto il concetto di "programmabile" all'interno delle tecnologie blockchain grazie agli *Smart Contracts*. Questi programmi implementano la logica di una normale applicazione secondo il paradigma object-oriented, con l'aggiunta della trasparenza e sicurezza del layer sottostante (la blockchain con il suo algoritmo di consenso). Il termine "smart contract" deriva proprio dal fatto che fungono da "contratto" per gli utilizzatori, definendo regole e registrando eventi direttamente sulla blockchain.

Su Algorand questo approccio è stato totalmente rivisitato tramite degli smart contract meno "smart" rispetto alla controparte Ethereum, ma con la peculiarità di essere eseguiti come delle semplici transazioni sottomesse alla rete e poi approvate. Non vi è dunque la presenza di una componente aggiuntiva adibita all'esecuzione degli smart contracts (come accade per Ethereum con la EVM) ma è il software core del nodo ad interpretare la logica che implementano.

Prendono il nome di *Algorand Smart Contract (ASC1)* e la loro funzione si limita all'autorizzazione di una data transazione. Possono pertanto essere considerati dei programmi booleani, con risultato true o false (successo o fallimento della transazione). Vedremo come in

realità questi strumenti apparentemente basici permettano di dar vita a sistemi abbastanza complessi grazie anche alla possibilità di combinarli con altri strumenti della blockchain.

ATTENZIONE: si parla di "autorizzazione" e non di "validazione". I due concetti sono infatti totalmente distinti: si parla di validazione in riferimento ai blocchi da aggiungere al ledger, mentre si parla di autorizzazione intesa come la verifica dei parametri di una transazione. Quando una transazione viene autorizzata non è ancora da considerarsi definitiva, deve infatti prima seguire l'intero iter previsto dall'algoritmo di consenso prima di essere inserita in un blocco e riportata definitivamente sul ledger.

Gli ASC1 si dividono in due categorie: *Stateful Smart Contracts* e *Stateless Smart Contracts*. Come si evince dalla nomenclatura i primi prevedono il concetto di "stato", mentre i secondi non presentano questa caratteristica. Vedremo questa distinzione più in dettaglio nei successivi paragrafi, ma concentriamoci prima sul linguaggio utilizzato per lo sviluppo degli Smart Contracts in Algorand, chiamato *TEAL*.

3.3.1 TEAL: Transaction Execution Approval Language

Il linguaggio utilizzato per lo sviluppo di Smart Contract su Algorand, chiamato *TEAL* [15], è un linguaggio stack-based, non Turing completo, che viene eseguito all'interno delle transazioni con il paradigma delle *Stack Machines*. I programmi TEAL vengono valutati in modo sequenziale (una istruzione per volta) tramite delle operazioni di push & pop di valori dallo stack. I valori rappresentabili all'interno dello stack sono di due tipi: `Uint64`, interi a 64bit senza segno, e `Bytes` (array di byte) entrambi manipolabili con una serie di operatori nativi del linguaggio. È possibile inoltre passare dei parametri allo smart contract (anch'essi aventi come tipo `Uint64` o `Bytes`) ed eseguire operazioni aritmetiche sfruttando uno *Scratch Space* temporaneo e locale alla singola esecuzione del contratto.

Dato che si tratta di programmi volti alla verifica ed autorizzazione di transazioni (da cui *Transaction Execution Approval Language*), la caratteristica principale di questi programmi è la possibilità di accedere a tutti i campi della transazione (o come vedremo di un gruppo di transazioni) ed effettuare dei controlli logici sui valori in essi contenuti. La struttura generale di un framework TEAL si presenta come mostrato in Figura 3.7 [16]:

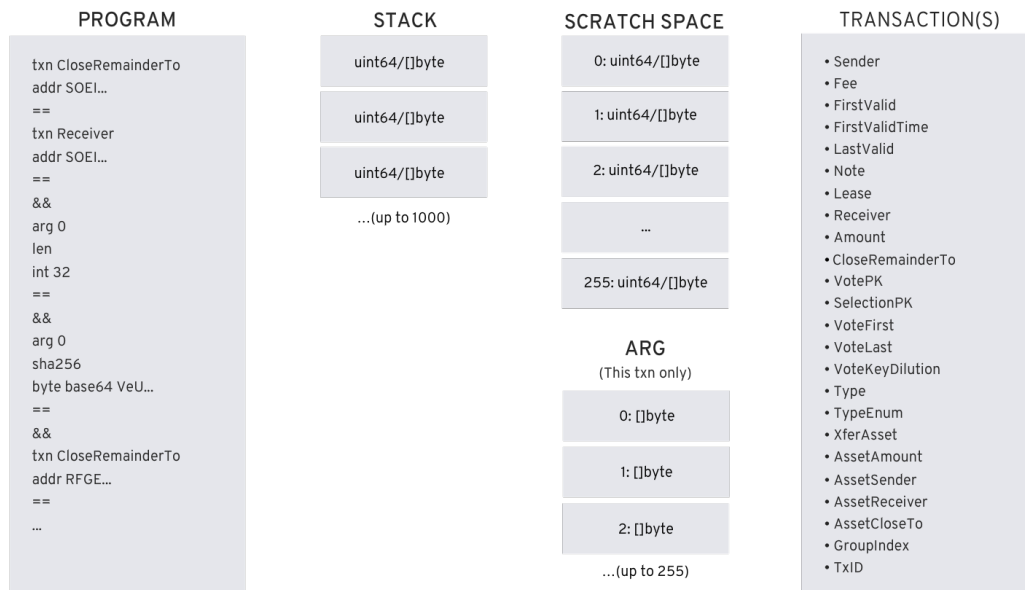


Figura 3.7: TEAL Architecture Overview

3.3.2 Stateful Smart Contracts

Questa tipologia di smart contract "vive" sulla blockchain ed è utilizzata per tenere traccia dello stato globale dell'applicazione nonché dello stato locale dell'utente che invoca il contratto. Anche in questo caso il tipo di dati mantenibili può essere di tipo Uint64 o Bytes, e la quantità di slot da riservare per la memorizzazione di questi parametri va definita in fase di deploy del contratto e non è modificabile nel tempo, se non con un update del contratto stesso. Nonostante le limitazioni di un linguaggio non Turing-completo come il TEAL e l'assenza di costrutti tipici dei linguaggi che invece lo sono (loops, branches ecc.), oltre che l'impossibilità di avere uno storage in stile data base associato al contratto, questi smart contract sono considerati delle vere e proprie applicazioni. Nel seguito della documentazione saranno pertanto riferiti con il nome di *Application Contract*².

La struttura di un application contract è composta di due parti: un *Approval Program* ed un *Clear State Program*. Le due parti del programma svolgono ruoli differenti e devono sempre essere presenti in ogni caso. L'approval program implementa tutti i flussi di controllo volti ad autorizzare la transazione (o il gruppo di transazioni), mentre il clear state program implementa la routine da eseguire nel momento in cui un account utilizzatore vuole annullare la sua sottoscrizione al contratto. Il modo con il quale si effettua una chiamata ad

²<https://developer.algorand.org/docs/features/asc1/stateful/>

una funzione del contratto è rappresentato da una particolare transazione di tipo *ApplicationCall*. Le chiamate all'applicazione sono quindi registrate sul ledger proprio come ogni altra transazione del sistema (transazioni di pagamento, trasferimenti di asset ecc.). Per poter essere invocato da un account, lo smart contract richiede che questo effettui prima una chiamata di *OptIn* con la quale esprime la sua volontà ad utilizzare l'applicazione. Allo stesso modo per recedere dal contratto l'account deve effettuare una chiamata di *CloseOut*. Un'altra funzionalità importante degli stateful smart contract è quella di poter recuperare (in sola lettura) lo stato globale di altri stateful smart contract associando all'applicazione principale, in fase di deploy, l'ID univoco di un'altra applicazione. Questa funzionalità permette di creare sistemi composti da contratti multipli sfruttando la possibilità di leggere gli uni lo stato globale degli altri e impiegare poi questi dati all'interno della logica del contratto. Tuttavia gli application contracts non possono invocare (attivare) applicazioni esterne, nè effettuare transazioni di alcun tipo [17].

3.3.3 Stateless Smart Contracts

La tipologia stateless differisce dagli application contracts sia per l'assenza del concetto di stato globale e/o locale, che per la modalità con cui questi contratti vengono invocati. Non si tratta infatti di applicazioni residenti sulla blockchain ma di programmi che vengono valutati in contemporanea alla sottomissione della transazione utilizzando la logica che implementano per valutare, ed eventualmente autorizzare, la transazione in oggetto. Sono da intendersi quindi come un flusso di controllo custom-built che precede i controlli standard effettuati dal software del nodo stesso. Questi contratti si dividono a loro volta in due modalità di funzionamento a seconda dello scopo per il quale sono progettati: *Contract Mode* e *Delegation Mode*. Entrambe le tipologie permettono di sostituire il meccanismo di firma diretta delle transazioni da parte dell'account che rilascia il contratto, permettendo a quest'ultimo di automatizzare il processo di firma/autorizzazione per specifici casi d'uso (come ad esempio l'automazione di un pagamento ricorrente a favore di un fornitore di servizi) delegando allo smart contract la facoltà di autorizzare o meno un determinato trasferimento.

- **Delegation Mode**

Uno stateless smart contract in *Delegation Mode*³ implementa la logica necessaria a valutare i parametri di una transazione, e in caso di successo vi appone la firma dell'account emittente. Vediamo brevemente i passi compiuti dalla creazione all'utilizzo del contratto, che prevedono l'interazione di due utenti *A* e *B*:

1. Per prima cosa *A* crea lo smart contract che implementa la logica che autorizzerà le transazioni emesse da *A*, appone la propria firma sullo smart contract e rilascia poi il contratto compilato alla controparte *B* che ne farà uso come detto prima, ad esempio, per effettuare un prelievo di fondi ricorrente in cambio dei servizi offerti. Il risultato ottenuto dalla compilazione del contratto firmato, è un oggetto che prende il nome di *Logic Signature*⁴;
2. Ogniqualvolta *B* vorrà effettuare una transazione di pagamento in suo favore da parte dell'account *A*, apporrà alla transazione la Logic Signature ottenuta al passo precedente, e invierà poi la transazione firmata alla rete;
3. A questo punto, prima che la transazione esca effettivamente dal nodo e venga propagata, viene prima controllata dalla logica presente nella Logic Signature: se la verifica va a buon fine, vi appone la firma di *A* sulla transazione, autorizzando così il pagamento in favore di *B* come se fosse stato *A* stesso ad emettere tale pagamento.

• Contract Mode

Analogamente a quanto detto per i *Delegation Contracts* anche in questa modalità la compilazione del contratto dà origine ad una *Logic Signature*. La differenza sta nel fatto che questi smart contract hanno un indirizzo proprio come un qualunque account della rete Algorand, e possono quindi ricevere pagamenti, sia in *Algos* (criptovaluta nativa di Algorand) sia in Algorand Standard Assets. Ciò che li rende diversi da un comune account è il modo in cui autorizzano le transazioni di pagamento *in uscita* dall'account.

Ad autorizzare queste operazioni è la logica del contratto stesso, che come nei *Delegation Contracts* verifica la transazione e vi appone la relativa Logic Signature. Smart contracts di questo tipo sono riferiti con il nome di *Contract Accounts*⁵ e possono esse-

³<https://developer.algorand.org/docs/features/asc1/stateless/modes/#delegated-approval>

⁴<https://developer.algorand.org/docs/features/asc1/stateless/modes/#logic-signatures>

⁵<https://developer.algorand.org/docs/features/asc1/stateless/modes/#contract-account>

re utilizzati per la creazione dei cosiddetti *escrow accounts*, ovvero accounts dai quali prelevare fondi con criteri prestabiliti. A differenza di un normale delegation contract però, qui non è necessario che la firma logica generata dal programma venga condivisa con coloro che intendono prelevare fondi, in quanto questa viene mantenuta ed utilizzata in modo automatico dal Contract Account stesso.

3.4 Algorand Standard Assets (ASA)

Gli Algorand Standard Asset⁶ forniscono un meccanismo standardizzato per la rappresentazione di dati arbitrari, direttamente sul layer-1 della blockchain. Si dividono in 2 categorie che sono, analogamente alla classificazione degli asset Ethereum visti in Sezione 2.4, i *fungible assets* e i *non fungible assets*. Su entrambe le tipologie valgono gli stessi concetti descritti per la controparte Ethereum, ma il modo in cui questi oggetti sono costruiti e gestiti sulla blockchain Algorand è sostanzialmente differente. A differenza di quanto detto per gli NFT su Ethereum, in questo contesto non vi è alcuna interfaccia applicativa che ne definisca le caratteristiche (i.e. ERC20/ERC721 su Ethereum), bensì viene delegato allo sviluppatore il compito di fissarne i parametri in fase di creazione per adattarli al meglio ai propri scopi. Ricordiamo che questi oggetti, come ogni altra feature presente al layer-1 della blockchain Algorand, vengono gestiti attraverso delle transazioni, inclusa la fase di creazione (coniazione).

Vediamo di quali campi si compone un ASA, il loro ruolo e quali opzioni prevedono:

Parametri **immutabili** (fissati in fase di creazione).

- CREATOR: indirizzo dell'account che crea l'asset (required);
- ASSETNAME: nome simbolico assegnato a questo asset dal creatore (recommended);
- UNITNAME: nome di una singola unità di questo asset (recommended);
- TOTAL: numero totale di unità dell'asset (required);
- DECIMALS: numero di cifre decimali. Definisce la divisibilità dell'asset e non può essere modificato dopo la creazione (required);

⁶ASA: <https://www.algorand.com/technology#ALGORAND-STANDARD-ASSETS>

- **DEFAULTFROZEN**: valore booleano che indica se le unità devono essere congelate di default dopo la creazione (required);
- **URL**: eventuale URL dal quale è possibile rinvenire ulteriori informazioni sull'asset (optional);
- **METADATAHASH**: valore hash di eventuali metadati rilevanti per questo asset (optional).

Vi sono inoltre 4 campi **mutevoli** (modificabili nel tempo), utilizzati per l'implementazione del protocollo *Role Based Asset Control (RBAC)*, ovvero una politica di gestione dell'asset basata proprio sui ruoli degli indirizzi degli account assegnati a questi campi, che definiscono quali autorizzazioni possiede ciascuno di essi su questo asset.

- **MANAGERADDRESS**: indirizzo dell'account proprietario di questo asset. L'asset manager è l'unico a poter modificare i valori di questo gruppo di parametri;
- **RESERVEADDRESS**: indirizzo dell'account che detiene le unità non ancora coniate (*non-minted*) di questo asset. Questo indirizzo non ha alcuna autorità specifica all'interno del protocollo ed è utilizzato solo per separare le unità coniate da quelle non coniate, in due account differenti;
- **FREEZEADDRESS**: indirizzo dell'account avente diritto a congelare le unità di questo asset (se non definito, l'opzione non è disponibile);
- **CLAWBACKADDRESS**: indirizzo dell'account avente diritto al recupero dei fondi (se il campo non è definito, l'opzione non è disponibile).

3.4.1 Fungible & Non-Fungible

La struttura appena vista è univoca per entrambe le tipologie di asset. Ciò che differenzia un fungible asset da un NFT su Algorand, sono i valori dei parametri **Total** e **Decimals**. Come detto in precedenza, un NFT è caratterizzato dalla sua unicità e indivisibilità, mentre un FT (fungible token) è disponibile in unità multiple e in genere divisibile in sotto unità. È quindi sufficiente settare i due parametri sopracitati nel seguente modo per ottenere il risultato desiderato:

- **Fungible**: $Total = n$, $Decimals = k$, con n arbitrario e $k = 0, \dots, 10$;

- **Non-Fungible:** Total = 1, Decimals = 0.

Ciascuna delle tipologie di asset sopra descritte si adatta meglio a specifici campi applicativi, come mostrato in figura 3.8 .

FUNGIBLE TOKENS	NON-FUNGIBLE TOKENS	RESTRICTED FUNGIBLE TOKENS	RESTRICTED NON-FUNGIBLE TOKENS
• In Game Points	• In Game Items	• Securities	• Real Estate
• Stable Coins	• Supply Chain	• Gov't Issued Fiat	• Ownership Registries
• Loyalty Points	• Real Estate	• Certifications	• Regulatory Certifications
• System Credits	• Identity		
• Cryptocurrencies	• Certifications		
	• Collectables		

Figura 3.8: Esempi applicativi ASA [**algorand**’asa]

Una importante peculiarità degli ASA è il meccanismo con il quale proteggono gli utenti da rischi e utilizzi illeciti degli stessi. Trattandosi di entità rappresentative di molteplici oggetti, fisici o digitali che siano, richiedono una esplicita sottoscrizione da parte dell’account all’asset stesso al fine di prevenire lo spam, la ricezione di asset sottoposti a tassazione o che rappresentano un danno alla reputazione dell’utente e/o dell’organizzazione a cui è associato l’account. Proprio come avviene per gli *Application Contract* trattati nella Sezione 3.3.2, anche per gli ASA è necessario effettuare una operazione preliminare di *OptIn* prima di potervi interagire.

3.5 Transazioni

Nelle sezioni precedenti è emerso l’approccio ”transaction oriented” che Algorand ha attuato nella gestione delle varie features implementate dal sistema. Questa è se vogliamo la caratteristica distintiva della blockchain in oggetto rispetto per esempio alla blockchain di Ethereum, che affronta lo sviluppo di sistemi costruiti su di essa con un approccio più ad alto livello (i.e. smart contract come applicazioni che usano un paradigma a oggetti, assets definiti da smart contracts).

Poter gestire tutto quanto al layer-1 attraverso delle semplici transazioni sottoposte alla

rete, richiede la presenza di parametri volti alla personalizzazione delle transazioni stesse che permettono di utilizzarle per eseguire specifici task. Le transazioni Algorand sono codificate in formato JSON e possiamo identificarne 4 categorie principali, ciascuna delle quali prevede ulteriori sottocategorie.

3.5.1 Struttura e tipologie

In questo paragrafo vengono prima mostrati i principali campi di una generica transazione (tabella 3.1) e segue poi una descrizione delle transazioni disponibili, suddivise per categorie, mostrando template a titolo di esempio per permettere al lettore di familiarizzare con la loro struttura.

Campo	Tipo	Codice	Descrizione
Fee	uint64	"fee"	Ammontare pagato dal mittente a titolo di commissione.
FirstValid	uint64	"fv"	Indica il round a partire dal quale la transazione è valida.
GenesisHash	[32]byte	"gh"	Hash del genesis block della rete cui la transazione è destinata (betaNet, testNet, mainNet).
LastValid	uint64	"lv"	Indica l'ultimo round di validità della transazione.
Sender	Address	"snd"	Indirizzo del mittente.
TxType	string	"type"	Indica il tipo di transazione.
Note	[]byte	"note"	Può contenere fino ad 1Kb di dati arbitrari.

Tabella 3.1: Principali campi comuni a tutte le tipologie di transazioni.

1. **Key Registration:** in Sezione 3.2 abbiamo parlato della possibilità di scegliere se un determinato account voglia partecipare all'algoritmo di consenso o meno, settando la sua modalità di funzionamento rispettivamente in *online* e *offline*. Questa imposita-

zione è modificabile attraverso una transazione di tipo KEYREG che definisce le due modalità in base ai parametri forniti alla transazione⁷.

```
{
  "txn": {
    "fee": 2000,
    "fv": 6002000,
    "gh": "SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/c0UJ0iI=",
    "lv": 6003000,
    "selkey": "X84ReKtMp+yfgmMCbbokVqeFFFrKQeFZKEXG89SXwm4=",
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXNMMARJHDCCLIHZU6TBE0C7XRSBG4",
    "type": "keyreg",
    "votefst": 6000000,
    "votekd": 1730,
    "votekey": "eXq34wzh2UIxCZaI1leALKyAvSz/+X0e0wqdHagM+bw=",
    "votelst": 9000000
  }
}
```

Figura 3.9: Transazione keyreg (online mode)

```
{
  "txn": {
    "fee": 1000,
    "fv": 7000000,
    "gh": "SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/c0UJ0iI=",
    "lv": 7001000,
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXNMMARJHDCCLIHZU6TBE0C7XRSBG4",
    "type": "keyreg"
  }
}
```

Figura 3.10: Transazione keyreg (offline mode)

2. **Payment Transactions:** una transazione di pagamento può essere utilizzata per due scopi differenti, o come una normale transazione di pagamento⁸, oppure per chiudere un account rimuovendolo dal ledger e trasferendo i fondi rimanenti ad un account esterno⁹ indicato nel campo "close" della transazione come mostrato in figura 3.12.
3. **Asset Transactions:** Le transazioni relative alla gestione degli asset si dividono in 3 sotto categorie relative rispettivamente alle operazioni di creazione/configurazione/-distruzione¹⁰ dell'asset, sottoscrizione/revoca/riconfigurazione¹¹ dell'asset, e conge-

⁷<https://developer.algorand.org/docs/features/transactions/#key-registration-transaction>

⁸<https://developer.algorand.org/docs/features/transactions/#payment-transaction>

⁹<https://developer.algorand.org/docs/features/transactions/#close-an-account>

¹⁰<https://developer.algorand.org/docs/features/transactions/#asset-configuration-transaction>

¹¹<https://developer.algorand.org/docs/features/transactions/#asset-transfer-transaction>

```
{
  "txn": {
    "amt": 5000000,
    "fee": 1000,
    "fv": 6000000,
    "gen": "mainnet-v1.0",
    "gh": "wGHE2Pwvd7S12BL5Fa0P20EGYesN73ktiC1qzkkit8=",
    "lv": 6001000,
    "note": "SGVsbG8gV29ybGQ=",
    "rcv": "GD64YIY3TWGDMCNPP553DZPPR6LDUSFQ0IJVFDPPXWEG3FV0JCCDBBH5A",
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
    "type": "pay"
  }
}
```

Figura 3.11: Transazione di pagamento standard

```
{
  "txn": {
    "close": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
    "fee": 1000,
    "fv": 4695599,
    "gen": "testnet-v1.0",
    "gh": "SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 4696599,
    "rcv": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
    "snd": "SYGHtA2DR5DYFWJE6D4T34P4AWGCG7JTNMY4VI6EDUVRMX7NG4KTA2WMDA",
    "type": "pay"
  }
}
```

Figura 3.12: Transazione di pagamento con opzione di chiusura dell'account

lamento¹² dell'asset. Queste transazioni vengono autorizzate secondo il protocollo RBAC descritto nella Sezione 3.4.

- Asset Configuration: la creazione di un asset tramite una transazione di tipo "acfg" (asset configuration) è caratterizzata dalla presenza di una struttura "apar" (asset parameters) all'interno della codifica Json della transazione stessa, come mostrato in Figura 3.13.

¹²<https://developer.algorand.org/docs/features/transactions/#asset-freeze-transaction>

```

{
  "txn": {
    "apar": {
      "am": "gXHjtDdtVpY7IKwJYsJWdCSrnUyRsX4jr3ihzQ2U9CQ=",
      "an": "My New Coin",
      "au": "developer.algorand.org",
      "c": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
      "dc": 2,
      "f": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
      "m": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
      "r": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
      "t": 5000000,
      "un": "MNC"
    },
    "fee": 1000,
    "fv": 6000000,
    "gh": "SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/c0UJ0iI=",
    "lv": 6001000,
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
    "type": "acfg"
  }
}

```

Figura 3.13: Creazione di un asset

Successivamente alla creazione è possibile modificare i parametri descritti in Sezione 3.4 tramite una asset configuration che ridefinisce questi campi all'interno della struttura "apar" come mostrato in figura 3.14.

```

{
  "txn": {
    "apar": {
      "c": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
      "f": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
      "m": "QC7XT7QU7X6IHNRJZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HVSQ",
      "r": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4"
    },
    "caid": 168103,
    "fee": 1000,
    "fv": 6002000,
    "gh": "SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/c0UJ0iI=",
    "lv": 6003000,
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
    "type": "acfg"
  }
}

```

Figura 3.14: Riconfigurazione di un asset

Vi è infine la possibilità di distruggere un asset precedentemente creato. Per fare ciò è necessario che l'asset in questione venga prima ritrasferito al creatore dell'asset, e solo a quel punto è possibile richiederne la distruzione con una transazione di tipo "acfg" priva della struttura "apar", come mostrato in figura 3.15.

```

{
  "txn": {
    "caid": 168103,
    "fee": 1000,
    "fv": 7000000,
    "gh": "SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/c0UJ0iI=",
    "lv": 7001000,
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXNMMARJHDCCCLIHZU6TBE0C7XRSBG4",
    "type": "acfg"
  }
}

```

Figura 3.15: Distruzione di un asset

- Asset Transfer: come già anticipato nella Sezione 3.4, per poter effettuare operazioni di trasferimento degli asset è necessario che l'account effettui prima la sottoscrizione all'asset. Questa operazione viene effettuata tramite una transazione di tipo "axfer" (asset transfer) caratterizzata dai due campi "arcv" (asset receiver) e "snd" (transaction sender) con valore coincidente, a indicare che chi sta effettuando la transazione autorizza la ricezione dell'asset sul proprio account. La struttura di questa particolare transazione è descritta in Figura 3.16.

```

{
  "txn": {
    "arcv": "QC7XT7QU7X6IHNJRZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HSVQ",
    "fee": 1000,
    "fv": 6631154,
    "gh": "SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/c0UJ0iI=",
    "lv": 6632154,
    "snd": "QC7XT7QU7X6IHNJRZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HSVQ",
    "type": "axfer",
    "xaid": 168103
  }
}

```

Figura 3.16: Sottoscrizione ad un asset

In modo analogo è possibile revocare tale autorizzazione con lo stesso tipo di transazione, specificando tra i parametri un asset sender (campo "asnd") associandogli l'indirizzo dell'account che vuole revocare la sua sottoscrizione, ed il relativo asset receiver (campo "arcv") per attuare il meccanismo di *clawback*, ovvero il trasferimento dei fondi verso un account esterno, previa distruzione delle unità.

```

{
  "txn": {
    "aamt": 500000,
    "arcv": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBE0C7XRSBG4",
    "asnd": "QC7XT7QU7X6IHNRJZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HSVQ",
    "fee": 1000,
    "fv": 7687457,
    "gh": "SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/cOUJ0iI=",
    "lv": 7688457,
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBE0C7XRSBG4",
    "type": "axfer",
    "xaid": 168103
  }
}

```

Figura 3.17: Revoca della sottoscrizione ad un asset con meccanismo di clawback

Il trasferimento vero e proprio di un asset presuppone che il ricevente della/e unità abbia prima effettuato la sottoscrizione all'asset con il meccanismo visto prima. La struttura di questa transazione è molto semplice e non prevede la definizione di parametri relativi all'asset oltre all'ammontare di unità da trasferire e l'identificativo dell'asset tramite il campo "xaid" (asset ID);

```

{
  "txn": {
    "aamt": 1000000,
    "arcv": "QC7XT7QU7X6IHNRJZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HSVQ",
    "fee": 3000,
    "fv": 7631196,
    "gh": "SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/cOUJ0iI=",
    "lv": 7632196,
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBE0C7XRSBG4",
    "type": "axfer",
    "xaid": 168103
  }
}

```

Figura 3.18: Trasferimento di un asset

- Asset Freeze: l'ultima transazione relativa alla gestione degli asset è quella di tipo "afrz" (asset freeze) che permette di congelare le unità al fine di renderle non trasferibili finchè non verranno scongelate. La costruzione della transazione prevede la definizione dei parametri relativi all'indirizzo dell'account impostato come "freeze address" (campo "fadd") al momento della creazione dell'asset (corrisponde con l'indirizzo di chi richiede il freeze, ovvero il mittente di questa transazione), e l'indirizzo dell'account al quale congelare le unità possedute.

```

{
  "txn": {
    "afrz": true,
    "fadd": "QC7XT7QU7X6IHNJRJZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HSVQ",
    "faid": 168103,
    "fee": 1000,
    "fv": 7687793,
    "gh": "SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/cOUJ0iI=",
    "lv": 7688793,
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBE0C7XRSBG4",
    "type": "afrz"
  }
}

```

Figura 3.19: Congelamento di un asset

Potrebbe sembrare un rischio quello di affidare ad un account esterno il potere di congelare le unità dell'asset, di proprietà di altri account. In realtà la possibilità di affidare ruoli diversi ad account diversi, si rivela uno strumento molto potente. Questa caratteristica degli ASA, infatti, può essere sfruttata per implementare politiche di sicurezza articolate all'interno della nostra applicazione. Inoltre come vedremo, queste impostazioni possono essere verificate in qualunque momento permettendo all'utente di tutelarsi da eventuali rischi.

Prima di procedere con la descrizione delle transazioni relative all'invocazione degli Stateful Smart Contract, riassumiamo per maggiore chiarezza le transazioni viste fino a questo punto, nella tabella 3.2.

Transazioni		
Gruppo	Tipologia	Descrizione
Key Registration	Online Registration	Registra l'account per la partecipazione al protocollo di consenso.
	Offline Registration	Revoca la partecipazione dell'account all'algoritmo di consenso.
Payment	Simple Payment	Effettua un trasferimento di Algo
	Account Closure	Chiude un account e trasferisce i fondi residui all'account di clawback.
Asset Configuration	Creation	Crea un nuovo asset (FT o NFT).
	Reconfiguration	Riconfigura i parametri mutevoli dell'asset.
	Destruction	Distrugge tutte le unità dell'asset.
Asset Transfer	Opt-in	Autorizza la ricezione di un asset.
	Revoke	Revoca l'autorizzazione a ricevere un asset.
	Simple Transfer	Trasferisce n unità dell'asset.
Asset Freeze	Freeze	Congela le unità di un asset.

Tabella 3.2: Riepilogo delle transazioni.

4. Application Transactions:

Quest'ultima transazione ha un ruolo diverso dalle precedenti categorie ed implementa il meccanismo con il quale gli utenti possono interagire con gli *Application Contracts* visti nella Sezione 3.3.2. Si tratta infatti di una singola transazione che svolge task diversi a seconda dei parametri impostati in fase di costruzione della transazione. Vediamo in linea di principio come funziona e quali operazioni permette di effettuare:

- **Creazione:** il deploy di uno stateful smart contract si effettua tramite una transazione di tipo *ApplicationCall* (campo `type="appl"`) alla quale bisogna allegare l'approval program ed il clear state program tramite i relativi campi della transazione, nonché eventuali applicazioni o asset esterni da utilizzare all'interno

dello smart contract. Questi due ultimi oggetti in particolare vengono utilizzati dal nodo che esegue lo smart contract per caricare in memoria i dati relativi ad essi, al fine di migliorare le prestazioni del sistema evitando di dover recuperare il loro stato a runtime ad ogni esecuzione. Inoltre, come già anticipato nella Sezione 3.3.2, bisogna dichiarare quali e quanti slot riservare all'applicazione per la memorizzazione di campi globali e locali. Il risultato di queste transazioni è un ID univoco assegnato all'applicazione dalla rete stessa, ed utilizzabile in seguito per riferire lo smart contract;

- **Sottoscrizione:** per poter interagire, o meglio invocare lo smart contract, bisogna prima effettuare una sorta di registrazione all'applicazione ("optin" nella terminologia di Algorand) che ha lo scopo di abilitare l'account richiedente all'utilizzo dell'applicazione stessa. Il risultato di questa chiamata setta l'ambiente locale per l'account in oggetto, all'interno del quale verranno memorizzati i campi previsti per quella specifica applicazione;
- **Revoca della sottoscrizione:** analogamente a quanto detto per gli ASA è possibile revocare la propria sottoscrizione ad un'applicazione, alla quale ci si era precedentemente registrati. Da notare che questa operazione non elimina le variabili di stato locali dell'utente, ma si limita solamente a marcarlo come utente non registrato;
- **Clear:** se invece si vuole sia revocare la sottoscrizione che eliminare il proprio stato locale relativo ad essa, bisogna effettuare una chiamata di tipo *Clear State*. Mentre tutte le altre operazioni innescano l'esecuzione della componente *Approval Program* del contratto, la clear state viene gestita dalla omonima componente *Clear State Program*, all'interno della quale è possibile implementare una logica di controllo che impedisce all'account di recedere definitivamente dall'applicazione qualora ci fossero degli obblighi, non ancora soddisfatti, nei confronti del sistema e/o di altri utenti;
- **Chiamata con argomenti:** l'operazione senza dubbio più potente effettuabile tramite una transazione di tipo *ApplicationCall* è l'invocazione con argomenti. All'interno dell'*approval program*, infatti, è possibile definire delle operazioni (sempre volte all'autorizzazione della transazione) personalizzate. Sono proprio queste a definire quali funzionalità mette a disposizione il contratto, in-

vocate specificando che la application call è di tipo "NoOp" tramite il relativo campo della transazione;

- **Update:** gli stateful ASC1 possono essere aggiornati durante la loro esistenza tramite una application call di tipo **UpdateApplication**, la quale è del tutto analoga all'operazione di deploy, con la differenza che va specificato l'application ID del contratto (ottenuto al momento del primo deploy);
- **Delete:** infine vi è la possibilità di eliminare dalla blockchain un'applicazione precedentemente creata tramite una application call di tipo *DeleteApplication*. Sia l'operazione di Update che di Delete seguono un flusso di controllo implementato nel *Approval Program* del contratto, al fine di tutelare gli utilizzatori dell'applicazione da intenzioni malevole da parte del gestore.

Riassumiamo nella tabella 3.3 tutti i campi presenti all'interno delle ApplicationCall:

Campo	Tipo	ID	Descrizione
ApplicationID	uint64	"apid"	ID dell'applicazione (vuoto per la creazione).
OnComplete	uint64	"apan"	Opzione di invocazione (rif. tabella 3.4).
Accounts	Address	"apat"	Lista di account accessibili dall'applicazione.
Approval Program	Address	"apap"	Logica dell'Approval Program.
Clear State Program	Address	"apsu"	Logica del Clear State Program.
App Arguments	byte[]	"apaa"	Lista di argomenti.
Foreign Apps	Address	"apfa"	Lista di applicazioni esterne.
Foreign Assets	Address	"apas"	Lista di asset accessibili dall'applicazione.
Global State Schema	StateSchema	"apgs"	Storage schema globale (rif. tabella 3.5).
Local State Schema	StateSchema	"apls"	Storage schema locale (rif. tabella 3.5).

Tabella 3.3: Campi di una Application Call.

Opzione	Descrizione
NoOp	Esegue solamente l'Approval Program senza effetti addizionali.
OptIn	Prima di eseguire l'Approval Program, alloca lo stato locale nell'account mittente.
CloseOut	Dopo aver eseguito l'Approval Program, pulisce lo stato locale dell'account mittente.
ClearState	Esegue il Clear State Program e pulisce lo stato locale dell'account mittente.
UpdateApplication	Dopo aver eseguito l'Approval Program, sovrascrive la logica dell'Approval Program corrente con quella nuova (fornita come parametro).
DeleteApplication	Dopo aver eseguito l'Approval Program, elimina l'applicazione rimuovendo lo stato globale (memorizzato nell'account del creatore).

Tabella 3.4: Opzioni di invocazione degli Application Contracts.

Campo	Tipo	ID	Descrizione
Number Ints	uint64	"nui"	Numero massimo di variabili di tipo Uint64 memorizzabili.
Number Byteslices	uint64	"nbs"	Numero massimo di variabili di tipo Bytes memorizzabili.

Tabella 3.5: Storage state schema.

3.5.2 Atomic Transfers

Quando si effettuano trasferimenti di fondi o asset in un contesto decentralizzato, vi è la necessità di gestire la correttezza delle parti. Si pensi ad esempio all'acquisto di un prodotto (i.e. un asset che rappresenti un quale valore o bene fisico) che richiede da una parte il pagamento dell'oggetto da parte dell'acquirente, e dall'altra l'effettivo trasferimento del bene. Si tratta di due transazioni distinte effettuate ciascuna dalla parte emittente, che avvengono

in momenti separati. Come ci si assicura che a seguito del pagamento il venditore trasferisca effettivamente l'asset mantenendo quindi l'accordo? E viceversa, come fa il venditore ad assicurarsi che l'acquirente paghi una volta ricevuto l'asset?

Un approccio utilizzato ad esempio da Ethereum per affrontare il problema è quello dei cosiddetti *hash time-locked contracts (HTLC)*¹³, ovvero applicazioni adibite alla gestione delle transazioni multi-parte come quella in esempio.

Algorand al contrario ha introdotto il concetto di *Atomic Transfer*[18] direttamente al layer-1. Il principio su cui si basano le atomic transfer è semplice: o tutte le transazioni in oggetto vanno a buon fine, oppure il fallimento di una singola transazione produce un fallimento dell'intero gruppo. Tornando all'esempio dell'acquisto di un asset consideriamo due utenti "Alice" e "Bob", rispettivamente nel ruolo di acquirente e venditore. Per condurre la loro trattativa con il meccanismo delle atomic transfer, devono sottoporre le due transazioni **senza apporvi la loro firma sopra**, combinare poi le due transazioni raggruppandole in una unica transazione, e solo a quel punto apporvi le loro firme. Ricordiamo infatti che apporre la propria firma su di una transazione emessa dal proprio account equivale ad autorizzare quella transazione, ma in questo caso vogliamo prima assicurarci che la controparte abbia rispettato i termini dell'accordo e solo a quel punto autorizzeremo la nostra transazione. La figura seguente riassume il processo appena descritto:

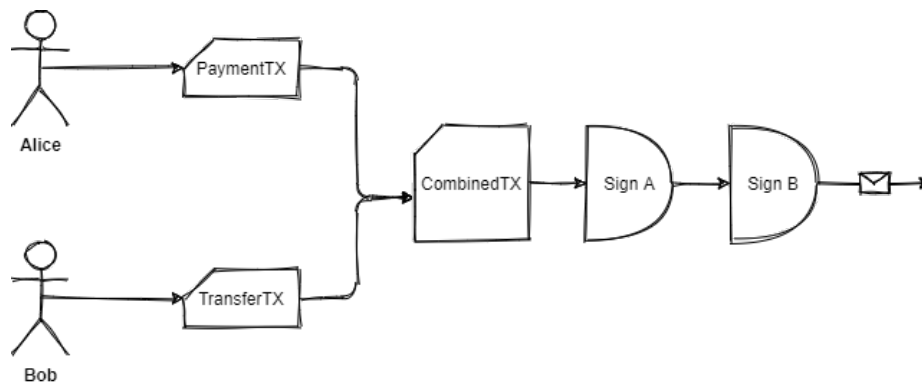


Figura 3.20: Atomic Transfer

Vedremo nel seguito di questo documento come il meccanismo delle atomic transfer, combinato con l'utilizzo di smart contract stateful e stateless, permetta di creare meccanismi piuttosto complessi con estrema efficienza.

¹³<https://smithandcrown.com/glossary/hashed-timelock-contract-htlc/>

3.6 Indexer V2

La blockchain di Algorand dispone di un potente strumento volto all'indicizzazione delle transazioni, che permette di effettuare query parametriche ad un database contenente le informazioni presenti sul ledger. Questo strumento, chiamato *Indexer*[19], nella sua attuale versione (v2) consiste in un processo che espone le Api REST per l'interazione con un database PostgreSQL. Il database in oggetto viene popolato dall'Indexer stesso, il quale preleva i dati dal processo *algod* (software core del nodo che implementa il protocollo di consenso, la propagazione delle transazioni ed il collegamento alla rete P2P) e li registra sotto forma di record nel DB con una struttura predefinita e non modificabile. La figura 3.21 mostra l'interazione tra le componenti appena descritte:

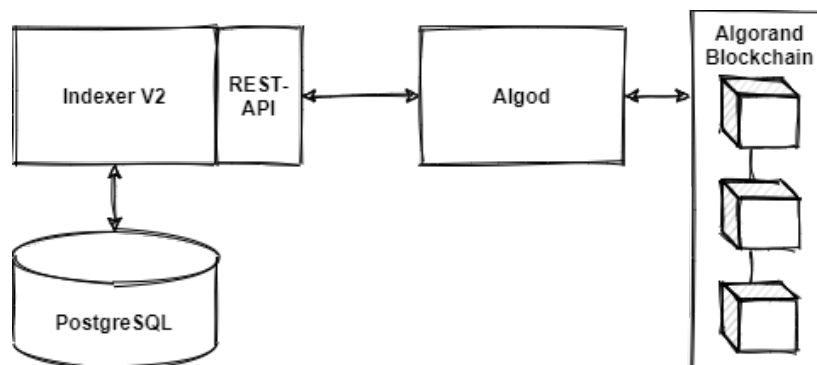


Figura 3.21: Algorand V2 Indexer

Le transazioni che vengono registrate all'interno del DB dall'Indexer sono tutte quelle presenti sul ledger fino al blocco dell'ultima sincronizzazione del nodo (una volta che il nodo ha concluso la sincronizzazione e scaricato i dati dell'intera blockchain, l'indexer conterrà di conseguenza tutte le transazioni registrate dal genesis block ad ora). Questa feature richiede che il nodo disponga di una grande quantità di memoria e che venga esplicitamente configurato per lavorare in modalità "archival"¹⁴.

Questo tool può essere istanziato sul proprio nodo al fine di utilizzarlo come strumento personale, da sfruttare ad esempio per il querying del ledger nelle proprie Dapps, oppure può essere utilizzato un service provider esterno che implementa l'indexer sul proprio nodo e lo espone alla rete mediante apposite Api proprietarie (e.g. AlgoExplorer¹⁵).

Questo tool rende la ricerca di informazioni molto efficiente e fruibile, in quanto non va a

¹⁴<https://developer.algorand.org/docs/run-a-node/setup/types/archival-mode>

¹⁵Algorand Blockchain Explorer: <https://algoexplorer.io/>

scandire tutti i blocchi del ledger ma sfrutta una loro rappresentazione compatta attraverso un comune server che può essere interrogato tramite semplici query parametriche. La tipologia di query effettuabili si divide in 4 macro categorie:

1. Transazioni;
2. Accounts;
3. Assets;
4. Applicazioni (Stateful ASC1).

La ricerca per ciascuna delle categorie sopra elencate può ulteriormente essere raffinata con l'aggiunta di filtri di ricerca molto specifici, come ad esempio il round (o round range) all'interno del quale vogliamo ricercare le nostre transazioni, il tipo specifico di transazione a cui siamo interessati, il tipo di firma che è stata apposta su quella transazione, e molti altri campi. I risultati delle query vengono restituiti in modo paginato, ovvero un gruppo per volta secondo i limiti descritti in figura 3.22. Insieme al gruppo di record restituiti dall'indexer, vi è anche un token detto *nexttoken*, che permette di proseguire la ricerca dal $n + 1$ -esimo record, passando questa informazione alla successiva invocazione dell'indexer.

Search Type	Maximum number of results per search
Transaction Search	1000
Account Search	100
Asset Search	100
Asset Balance Search	1000

Figura 3.22: Numero di record restituiti per ricerca

Come vedremo nel capitolo 4 questo strumento può essere sfruttato per eseguire query molto specifiche ed utilizzare la blockchain stessa come uno storage distribuito.

Capitolo 4

AlgoKitties

In questo capitolo vedremo applicati i concetti e gli strumenti descritti sino ad ora attraverso un'applicazione pratica su Algorand. Il caso d'uso proposto si basa sulla nota applicazione *CryptoKitties* [20] lanciata ufficialmente sulla rete Ethereum il 28 Novembre 2017. Si tratta di una game-application decentralizzata sviluppata su blockchain Ethereum tramite smart contracts, che intende avvicinare l'utente al mondo delle blockchain. Il gioco illustra le funzionalità della blockchain tramite la creazione, acquisto e vendita di collezionabili rappresentati graficamente da modelli 2D di gatti dalla natura puramente artistica.

Nella Sezione 4.1 verrà descritta più dettagliatamente la natura ed il funzionamento dell'applicazione *CryptoKitties*. Segue in Sezione 4.2 una descrizione del porting di *Cryptokitties* su Algorand, delle scelte implementative effettuate, e del funzionamento dell'applicazione riprogettata come oggetto di questo tirocinio.

4.1 Ethereum Cryptokitties

CryptoKitties è una game application sviluppata con un obiettivo ben preciso[20], ovvero quello di creare un caso d'uso reale che mettesse in risalto le potenzialità delle blockchain, fornendo all'utente una consapevolezza di quali siano i campi applicativi di tali sistemi, oltre a quello finanziario. Per raggiungere questo obiettivo i fondatori di *CryptoKitties* hanno pensato di creare qualcosa che fosse alla portata di tutti in termini di fruibilità e al contempo educasse gli stessi a comprendere più a fondo la tecnologia sottostante, fino ad allora percepita co-



me uno strumento rivolto al mercato finanziario. L'applicazione si basa sull'acquisto di collezionabili, in forma di gattini digitali, *kittens*, dalle caratteristiche uniche, che andranno a formare una collezione personale e unica per ciascun utente. Vediamo quali funzionalità implementa il sistema di riferimento:

- **Creazione dei kitten:** la creazione effettiva di un kitten avviene tramite un algoritmo di generazione segreto, con il quale il gestore del sistema (i.e. il team di Crypto-Kitties) dà vita ad esemplari unici, che successivamente si traducono in asset (NFT) della blockchain Ethereum;
- **Aste di vendita:** i collezionabili vengono immessi nel mercato (o shop) attraverso delle aste dalla durata prefissata, durante la quale gli utenti possono piazzare la loro offerta per un particolare gattino, partendo da un prezzo massimo fissato e procedendo al ribasso. La scelta delle aste al ribasso scaturisce dal fatto che con una normale asta inglese (prezzo minimo di partenza, e si procede al rialzo) gli utenti avrebbero dovuto piazzare continuamente delle offerte attraverso delle transazioni, il che si traduce in costi di gestione eccessivi per il pagamento delle commissioni necessarie per effettuare il mining di quelle transazioni. Affronteremo questo topic nel Capitolo ?? dedicato al confronto delle due implementazioni.

In questo modo invece l'utente sceglie il prezzo che intende pagare per quel kitten e si aggiudica la vittoria colui che giunge per primo al prezzo target (ovvero chi ha offerto di più);

- **Accoppiamento:** ogni kitten ha la capacità di accoppiarsi insieme ad un altro kitten al fine di generare un nuovo esemplare ottenuto a partire dal codice genetico dei genitori. Ciascun kitten, infatti, è caratterizzato da una stringa genetica (il genoma) che determina le caratteristiche fisiche dell'esemplare cui è associato.

I kitten sono implementati sulla rete Ethereum sotto forma di NFT secondo lo standard ERC721 già citato nella Sezione 2.4.

4.2 Implementazione su Algorand

La scelta di questo caso d'uso è scaturita principalmente dalla necessità di provare sul campo quali fossero le potenzialità di Algorand, attraverso un proof of concept che sfruttasse alcune delle features principali della blockchain. Sulla base di queste esigenze è stato sviluppato un porting che riporta le principali funzionalità del sistema di origine Cryptokitties, sul nuovo sistema oggetto di studio, Algorand. A tal proposito è stata tralasciata la componente grafica della game application, concentrandosi invece su tutte quelle funzionalità che rendono possibile la creazione ed il management degli asset, nonché il trading degli stessi. L'applicazione è stata rinominata *Algokitties*.

Essendo Ethereum ed Algorand due blockchain che adottano paradigmi molto differenti, è stato necessario riprogettare integralmente l'applicazione originale CryptoKitties per adattarla al nuovo sistema. L'implementazione originale di questo gioco, infatti, può vantare un insieme di strumenti molto potenti che non sono presenti su Algorand. Gli smart contracts di Ethereum portano con sé tutti i vantaggi del paradigma a oggetti, nonché l'interazione tra essi e la possibilità di essere innescati da eventi generati anche da altri smart contract. Viceversa, su Algorand gli smart contract non dispongono di costrutti iterativi, strutture dati articolate, o altre features tipiche di un linguaggio Turing completo, né possono essere innescati da eventi o generarne di propri.

In conclusione, l'implementazione Ethereum sfrutta le funzionalità object-oriented degli smart contracts, i quali una volta invocati generano degli eventi e dei risultati che saranno poi riportati sul ledger distribuito. Algorand invece è una tecnologia "transaction oriented" pura, nel senso che l'unico modo per registrare dei dati sul suo ledger è quello di effettuare delle transazioni, mentre gli smart contracts vengono utilizzati solamente per autorizzare tali transazioni con una logica appositamente personalizzata dallo sviluppatore. Come vedremo però, per superare le limitazioni dello stato attuale di questa tecnologia è sufficiente immaginare ciascuna funzionalità della game application come un insieme di transazioni strutturate secondo una logica ben precisa: che segua delle regole uniche per tutti i player, trasparenti in quanto riportate sulla blockchain tramite gli smart contract, e sicure in quanto autorizzate e registrate sul ledger.

4.2.1 Strumenti di sviluppo

Per lo sviluppo dell'applicativo è stata utilizzata una rete locale, istanziando un'immagine Docker¹ sulla quale sono in esecuzione i servizi del nodo Algorand, tra cui il core-software "algod" ed il servizio di indicizzazione del ledger, nonché il DB PostgreSQL a cui si collega. Di seguito sono riportati i componenti software di un nodo Algorand:

- **Algod**: processo principale, implementa il concetto di "nodo" Algorand e gestisce l'interfacciamento con la blockchain, invia le transazioni alla rete ed esegue il protocollo di consenso;
- **Indexer**: strumento di indicizzazione che si occupa della popolazione e interrogazione di un DB PostgreSQL contenente le informazioni relative alle transazioni registrate sul ledger;
- **Kmd**: *Key Management Deamon*, gestore dei wallets/accounts presenti nel nodo e delle rispettive chiavi associate ad essi. Questo tool è integrato nel software algod.

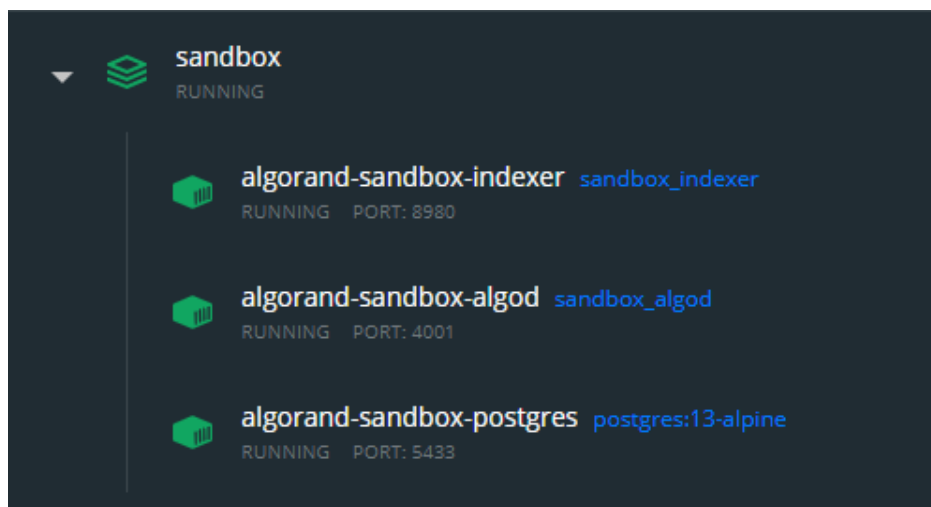


Figura 4.1: Container Docker del Nodo Algorand

I tre elementi appena descritti costituiscono complessivamente il software del nodo, ed espongono le Api REST, ciascuno su una porta differente, tramite le quali potervi interagire.

¹<https://github.com/algorand/sandbox>

4.2.2 Scelte implementative

Il sistema sviluppato consta di due applicazioni Java, una Server ed una Client, che implementano la comunicazione con i servizi sopra citati e che fungono da interfaccia per l'interazione con la blockchain Algorand. Sulla blockchain, invece, risiedono gli Stateful ASC1 che implementano la logica di autorizzazione delle varie operazioni implementate. La Figura 4.2 mostra l'architettura di Algokitties, mentre la Figura 4.3 il diagramma delle classi Java.

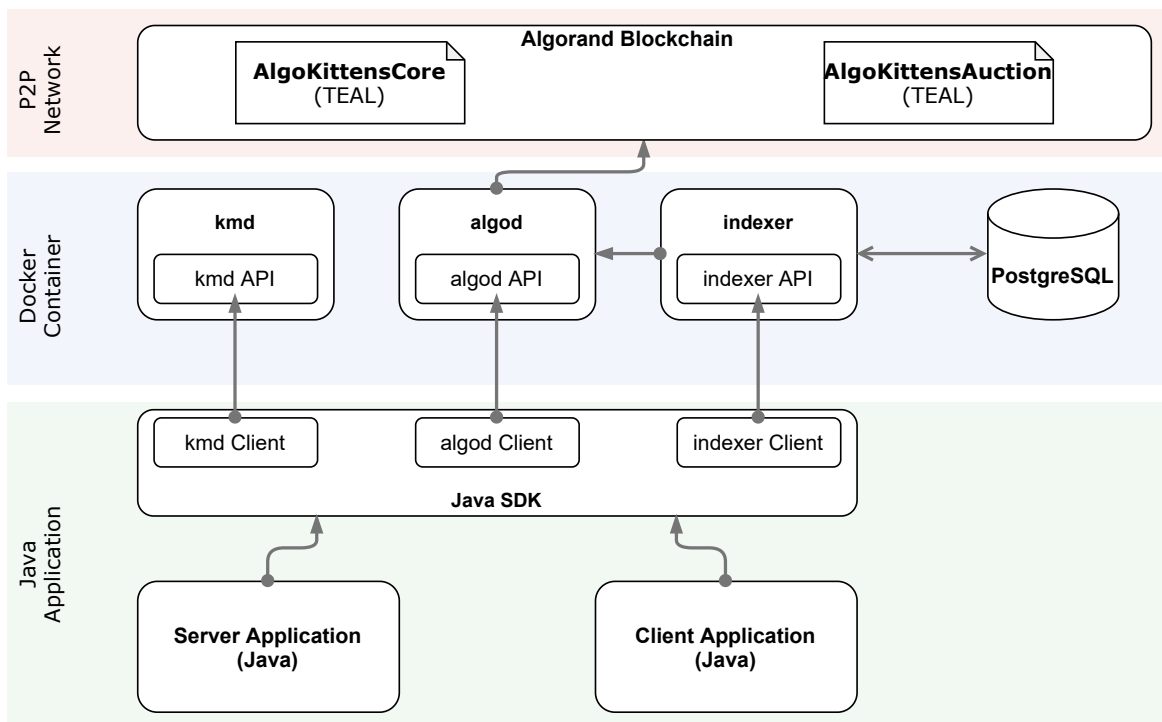


Figura 4.2: Architettura sistema Algokitties

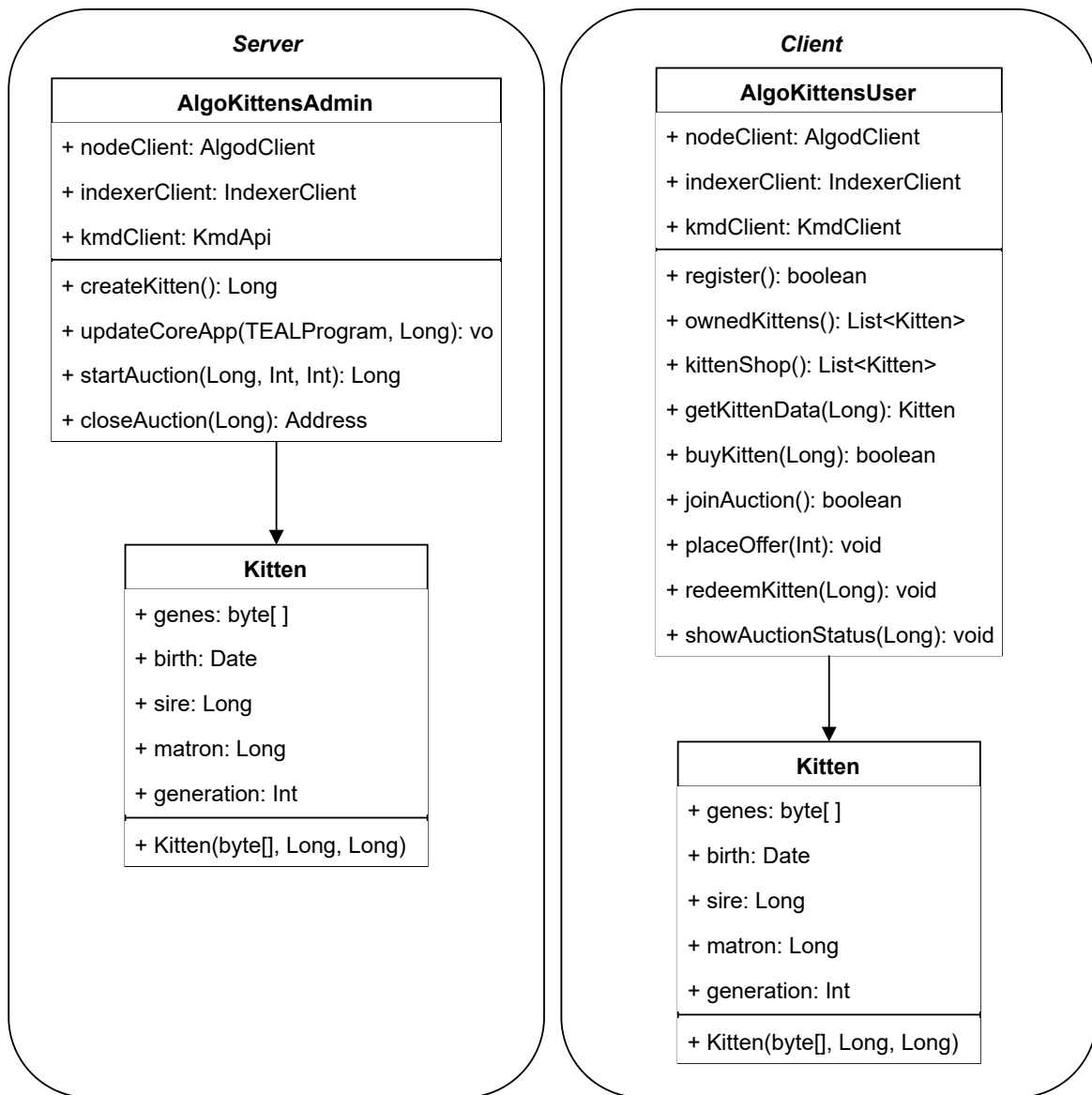


Figura 4.3: Diagramma delle classi delle Java Application

Gli Smart Contracts che implementano la logica di approvazione delle operazioni implementate da Algokitties, sono stati sviluppati utilizzando come linguaggio di programmazione Python e sfruttando la libreria PyTeal [21] che implementa il binding tra il linguaggio Python ed il linguaggio nativo degli smart contracts, ovvero TEAL. Gli smart contracts sviluppati sono:

- **AlgoKittensCore:** Stateful ASC1 che implementa la logica di approvazione delle transazioni relative alla gestione degli utenti, e la logica necessaria a gestire le aste;
- **AlgoKittensAuction:** Stateful ASC1 che implementa la logica di approvazione delle transazioni relative alle funzionalità delle aste di vendita, tra cui la partecipazio-

ne all'asta ed il piazzamento di un'offerta e le informazioni sull'asset in vendita e sull'eventuale vincitore ad asta conclusa;

- **SaleDelegation:** Stateless ASC1 in Delegation Mode che implementa la logica di autorizzazione per le operazioni di acquisto di un kitten;
- **RedeemDelegation:** Stateless ASC1 in Delegation Mode che implementa la logica di autorizzazione per le operazioni di riscossione del kitten da parte del vincitore dell'asta.

I termini "client" e "server" sono utilizzati per distinguere le due applicazioni che implementano rispettivamente le funzionalità per l'utente e quelle per l'amministratore del sistema. Di fatto non vi è alcuna comunicazione diretta tra le due entità software e non si tratta quindi di un paradigma client-server effettivo, proprio a voler focalizzare l'attenzione sulla tecnologia blockchain e su come sia possibile eseguire i vari task interagendo solamente con essa.

Tuttavia alcune funzionalità richiedono lo scambio delle LogicSignatures, viste in Sezione 3.3.3, volte ad autorizzare transazioni di acquisto: questo scambio viene effettuato attraverso la creazione di un file con estensione *.lsig* da parte del venditore, ed il successivo download del file da parte del compratore cui è rivolto. In un futuro sviluppo questo sistema verrà sostituito da una web application che permetterà una comunicazione diretta tra gli utenti della community di gioco, incluso lo scambio delle LogicSignatures.

4.2.3 Server Application

La server application espone tutte le funzionalità necessarie per il management dell'ecosistema Algokitties, dal deploy degli Application Contracts alla creazione dei collezionabili. Vediamo nel dettaglio le funzionalità implementate ed i relativi smart contracts che autorizzano le transazioni associate:

- **Deploy application**

Al primo avvio, l'applicazione Server effettua il deploy del contratto principale "AlgoKittensCore", tramite una transazione di tipo *ApplicationCall*. Da quel momento il sistema può considerarsi avviato e si possono cominciare ad effettuare tutte le operazioni lato amministratore e lato utente che il contratto prevede, tra cui la gestione degli utenti e la creazione di aste;

- **Update Application**

In modo analogo al deploy dell'applicazione, si effettua anche l'update della stessa mediante una *ApplicationCall* che ne va a sostituire la logica delle componenti *Approval Program* e *Clear State Program*. Per garantire che gli aggiornamenti alla core application non compromettano la sicurezza del sistema, lo smart contract tiene traccia della sua attuale versione nella variabile "Version" del suo stato globale, ed in modo analogo gli utenti mantengono l'informazione relativa alla versione del contratto al momento della loro sottoscrizione tramite una variabile "Version" all'interno del loro stato locale. Ciascuna delle operazioni effettuate dall'utente prevedono un check preliminare volto ad accertare la consistenza tra i due valori sopra citati;

- **Create Kitten**

La funzionalità senza dubbio più importante è quella che permette di creare nuovi esemplari di kitten da immettere nello shop di AlgoKitties. Questa operazione consta di una transazione di tipo "AssetConfig" finalizzata alla creazione di un non-fungible token associato all'oggetto *Kitten* creato localmente dall'applicazione Java.

```
Choose an option: 1
Sending transaction...
Transaction submitted, id: NGSJYTN72QKXTEIFRS5KI5PJW0ZNPJY06V0BSAA2LCEJLN6DZ6RA
Transaction NGSJYTN72QKXTEIFRS5KI5PJW0ZNPJY06V0BSAA2LCEJLN6DZ6RA confirmed in round 6451
Asset-id: 84
Created new kitten with id: 84
Genome string: 0cGrRYLSMz6X6i1i3PCWz2EFYosdw1VuIq+B7JHM2Sw=
```

Figura 4.4: Creazione di un Kitten

In Figura 4.4 è mostrato l'esito della transazione che ha appena creato un nuovo asset sulla blockchain, associandogli un id univoco (come per gli application contracts) che lo identifica.

L'associazione tra l'asset e l'oggetto Kitten generato dal server, viene effettuata sfruttando il campo *note* della transazione. L'oggetto Kitten viene serializzato in formato Json, ne vengono estratti i bytes ed inseriti all'interno di *note*. In questo modo il kitten è direttamente contenuto nell'asset sulla blockchain e non può essere modificato nel tempo. Lo schema seguente riassume il processo appena descritto:

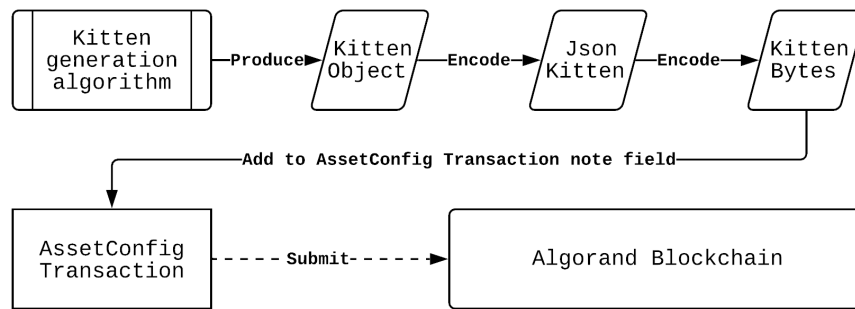


Figura 4.5: Caricamento di un Kitten sulla blockchain Algorand

Questo approccio permette di utilizzare la blockchain come un vero e proprio database distribuito, sfruttando il servizio Indexer per recuperare la transazione che ha creato l'asset, ma anche i dati contenuti nel campo *note* che potranno poi essere deserializzati a formare il Kitten.

- **Start Auction**

Come nel sistema originale Cryptokitties, anche in questa implementazione la vendita degli asset avviene principalmente attraverso un sistema di aste. L'amministratore crea un'asta specifica per ogni Kitten attraverso il deploy di un contratto "AlgoKittensAuction" tramite il quale ne definisce pubblicamente i parametri (prezzo iniziale, durata, ID del Kitten ecc.). Segue poi un update dello stato globale del contratto AlgoKittensCore che servirà da notifica (asincrona) per gli utenti, del fatto che un'asta è stata avviata. Previo deploy del contratto, l'amministratore del sistema (attuale proprietario del Kitten) congela l'asset in questione in modo da marcarlo come "non trasferibile" per tutta la durata dell'asta;

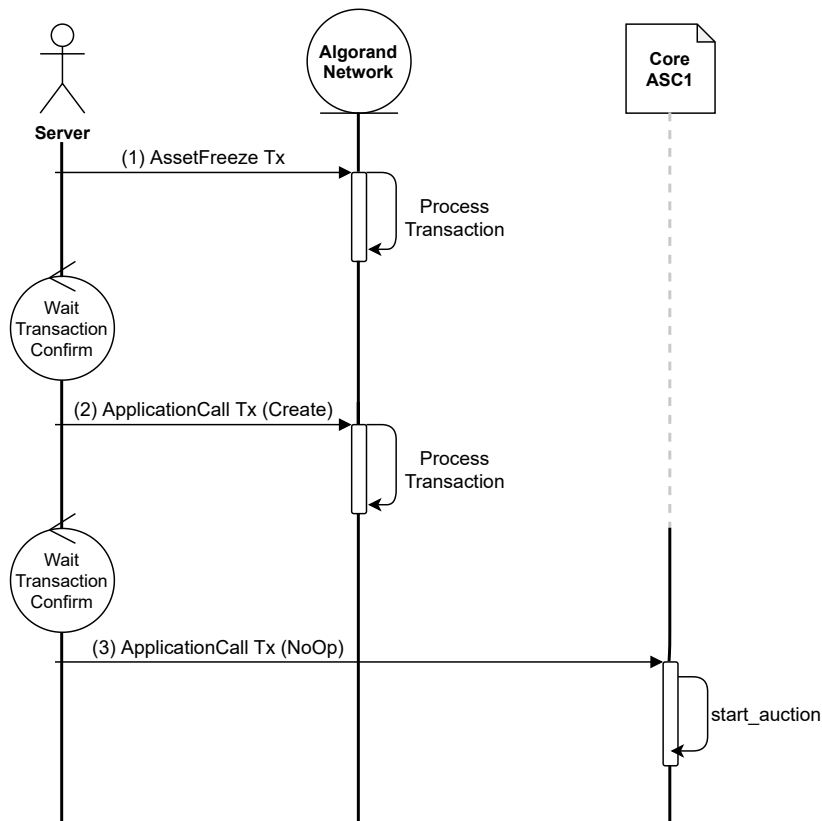


Figura 4.6: Schema semplificato dell'avvio di un'asta

```

1 start_auction = Seq([
2     # check issuer is admin
3     Assert(Txn.sender() == App.globalGet(Bytes("Admin"))),
4     Assert(Txn.application_args.length() == Int(3)),
5     # notify new auction is open
6     App.globalPut(Bytes("AuctionOpen"), Int(1)),
7     # put auction id in global state
8     App.globalPut(Bytes("AuctionID"), Btoi(Txn.application_args[1])),
9     # put asset id in global state
10    App.globalPut(Bytes("KittenID"), Btoi(Txn.application_args[2])),
11    Return(Int(1))
12 ])
  
```

Codice 4.1: Core contract start_auction

• Stop Auction

Una volta scaduto il termine per l'asta in corso, è necessario che questa venga chiusa eleggendo l'eventuale vincitore e preparando una LogicSig che permetterà al vincitore di riscattare l'asset, e scongelando l'asset per permetterne il trasferimento. Il meccanismo di congelamento e scongelamento previene che l'asset possa essere acquistato al di fuori dell'asta. Analogamente a quanto detto per l'avvio dell'asta, anche

alla sua chiusura viene effettuata una chiamata al contratto AlgoKittensCore volta a rimuovere le informazioni relative all'asta appena conclusa.

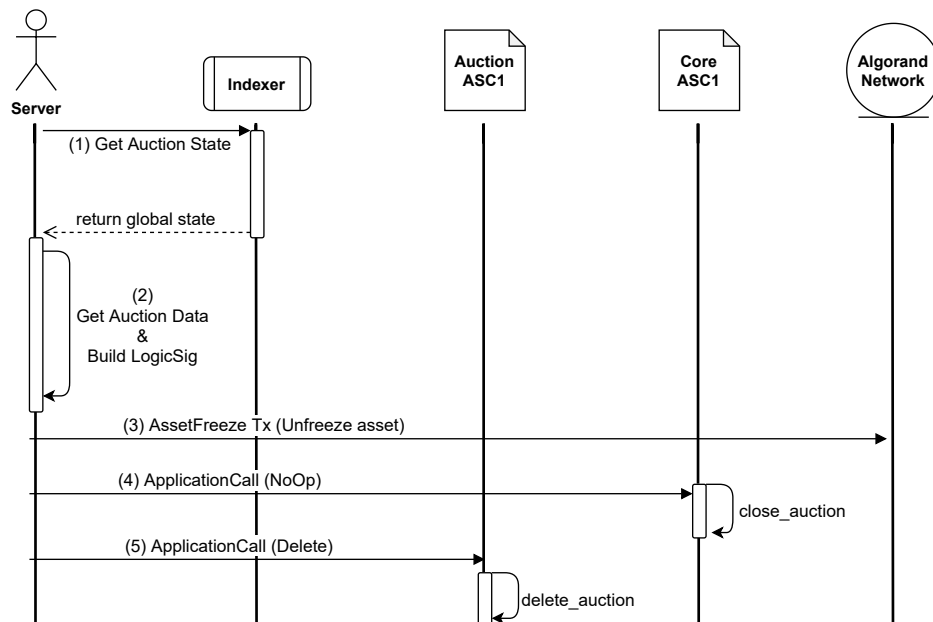


Figura 4.7: Schema semplificato della chiusura di un asta

Alla fine del processo descritto in Figura 4.7 l'asset sarà nuovamente scongelato e l'utente potrà richiederne il trasferimento. Il modo in cui viene gestito il trasferimento sarà descritto nella sezione seguente.

4.2.4 Client Application

La client application espone le funzionalità elencate di seguito all'utente:

- **Register**

La registrazione degli utenti avviene mediante sottoscrizione da parte dell'account al contratto AlgoKittensCore mediante una transazione di tipo ApplicationCall in modalità "OptIn". Questa transazione attiverà la funzione "on_register" del contratto, che setterà lo stato locale dell'account utente impostando due variabili di controllo, come mostrato nel codice seguente:

```

1 on_register = Seq([
2     # inizializza lo stato locale dell'utente
3     App.localPut(Int(0), Bytes("Registered"), Int(1)),
4     App.localPut(Int(0), Bytes("Version"), App.globalGet(Bytes("Version"))),

```



```

5   Return (Int (1))
6 ])
```

Codice 4.2: Core contract on_register

La variabile di stato "Registered" indicherà al contratto "AlgoKittensAuction" che l'utente è registrato al sistema e pertanto autorizzato a partecipare all'asta, mentre la variabile "Version" servirà per eseguire il check descritto in precedenza. Complessivamente il processo di registrazione può essere riassunto nello schema 4.8;

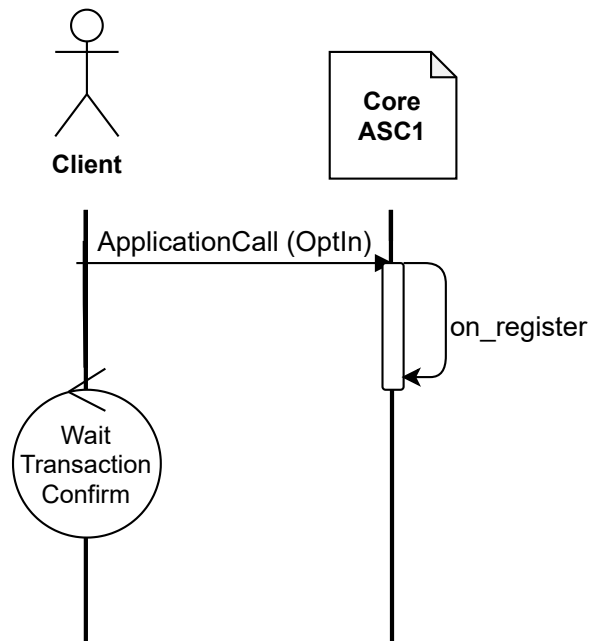


Figura 4.8: Registrazione utente

Una volta registrati, si potranno effettuare varie operazioni relative al trading dei Kitten, sia tramite acquisto diretto che tramite le aste di vendita;

- **Kitten Shop**

Il metodo *kittenShop* dell'applicativo Client permette di invocare il servizio Indexer per recuperare tutti gli asset (Kitten) attualmente in vendita, tramite query parametriche al DB PostgreSQL, ricercando tutti gli asset creati dall'account amministratore che siano di proprietà del suo account in quantità non nulla. La richiesta ritornerà una lista di tutti gli asset che rispettino queste condizioni e ne mostrerà l'ID associato, da utilizzare per effettuare l'acquisto, ottenendo un risultato simile a quello mostrato in Figura 4.9 .

```
----- 'Starchild' -----  
Kitten ID: 48  
Birth:   Mon Apr 05 17:08:37 CEST 2021  
Generation: 0  
Genetic String: [B@4f0100a7  
----- 'Boris' -----  
Kitten ID: 49  
Birth:   Mon Apr 05 17:09:17 CEST 2021  
Generation: 0  
Genetic String: [B@3cdf2c61  
----- 'Abby' -----  
Kitten ID: 50  
Birth:   Mon Apr 05 17:09:30 CEST 2021  
Generation: 0  
Genetic String: [B@13ad5cd3
```

Figura 4.9: Visualizzazione degli asset disponibili nello shop

- **Buy Kitten**

L'acquisto dei Kitten è implementato sfruttando il meccanismo delle Atomic Transfer e delle LogicSig, quest'ultima nella fattispecie è contenuta nello Stateless ASC1 "SaleDelegation". Questo smart contract implementa la logica necessaria a garantire che l'intero processo, dal pagamento alla ricezione dell'asset, avvenga in modo atomico e inscindibile. Per fare ciò il client esegue i passi indicati nello schema seguente:

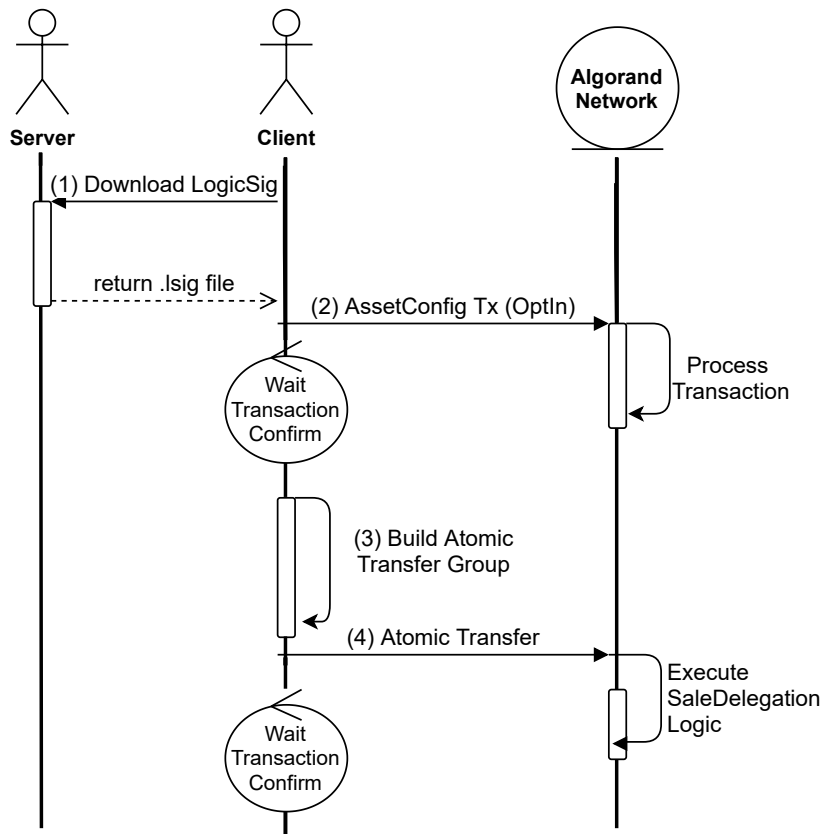


Figura 4.10: Acquisto di un Kitten

L'atomic transfer che include le transazioni di pagamento, trasferimento, e riconfigurazione dell'asset verrà autorizzata dalla logica del contratto "SaleDelegation" riportata di seguito:

```

1 is_complete_group = ( Global.group_size() == Int(3) )
2 valid_payment = And(
3   Gtxn[0].type_enum() == TxnType.Payment ,
4   Gtxn[0].amount() == Int(asset_price) ,
5   Gtxn[0].receiver() == Addr(seller_address)
6 )
7 valid_xfer = And(
8   Gtxn[1].type_enum() == TxnType.AssetTransfer ,
9   Gtxn[1].asset_receiver() == Gtxn[0].sender()
10 )
11 valid_cfg = And(
12   Gtxn[2].type_enum() == TxnType.AssetConfig ,
13   Gtxn[2].config_asset_manager() == Gtxn[0].sender()
14 )
15 validation = And(
16   is_complete_group , # verifica che tutte e tre le transazioni siano presenti
17   valid_payment ,     # check dei parametri per il pagamento
18   valid_xfer ,        # check dei parametri per il trasferimento dell'asset
  
```

```

19  valid_cfg      # check dei parametri per la riconfigurazione dell'asset
20 )

```

Codice 4.3: Sale Delegation Logic

• Join Auction

Il secondo metodo per acquistare dei Kitten è quello di aggiudicarseli durante un'asta indetta dall'amministratore con il metodo visto prima. Per poter partecipare ad un'asta l'utente deve prima effettuare la sottoscrizione al relativo contratto, mediante una semplice ApplicationCall di tipo "OptIn". Successivamente potrà, per tutta la durata dell'asta, inviare delle offerte e visualizzare lo status attuale (offerta più alta fino ad ora, round residui, asset in oggetto ecc.).

```

1  is_registered_user = (App.localGetEx(Int(0), coreID, Bytes("Registered")))
2  user_version = (App.localGetEx(Int(0), coreID, Bytes("Version")))
3  core_version = (App.globalGetEx(coreID, Bytes("Version")))
4  join_auction = Seq([
5      Assert(Not(is_owner)), # verifica che non sia il proprietario
6      is_registered_user,
7      user_version,
8      core_version,
9      # verifica che sia un utente registrato alla CoreApp
10     Assert(is_registered_user.hasValue()),
11     Assert(is_registered_user.value() == Int(1)),
12     Assert(user_version.hasValue()),
13     Assert(core_version.hasValue()),
14     # verifica che le versioni coincidano
15     Assert(core_version.value() == user_version.value()),
16     # aggiorna lo stato locale dell'utente
17     App.localPut(Int(0), Bytes("JoinedAuction"), Int(1)),
18     App.localPut(Int(0), Bytes("PlacedOffer"), Int(0)),
19     Return(Int(1))
20 ])

```

Codice 4.4: Auction contract join_auction logic

• Place Offer

Una volta effettuata la join all'Auction contract è possibile effettuare delle offerte tramite ApplicationCall al contratto stesso, il quale valuterà la transazione attraverso la logica seguente:

```

1  place_offer = Seq([
2      Assert(Not(is_owner)),
3      # verifica che l'asta non sia scaduta

```

```

4   Assert(Global.round() <= App.globalGet(Bytes("AuctionEnd"))),
5   # verifica che l'utente abbia regolarmente acceduto all'asta
6   Assert(App.localGet(Int(0), Bytes("JoinedAuction")) == Int(1)),
7   Assert(Txn.application_args.length() == Int(2)),
8   # verifica che l'offerta non sia inferiore al prezzo attuale
9   Assert(Btoi(Txn.application_args[1]) > App.globalGet(Bytes("CurrentPrice"))),
10  # in caso positivo, accetta la transazione e aggiorna lo stato dell'asta
11  App.globalPut(Bytes("Candidate"), Txn.sender()),
12  App.globalPut(Bytes("CurrentPrice"), Btoi(Txn.application_args[1])),
13  App.localPut(Int(0), Bytes("PlacedOffer"), Int(1)),
14  Return(Int(1))
15 ])
```

Codice 4.5: Auction contract place_offer logic

La variabile globale "Candidate" viene aggiornata con l'indirizzo dell'utente attualmente in testa e viene utilizzato, in fase di chiusura dell'asta, da parte dell'amministratore per constatare l'effettivo vincitore dell'asta e costruire la firma logica che gli servirà per riscuotere l'asset.

• Redeem Kitten

La riscossione dell'asset aggiudicatosi durante l'asta da parte di un utente, avviene con lo stesso meccanismo utilizzato per l'acquisto di un Kitten dallo shop, ovvero sfruttando i trasferimenti atomici. L'utente che ad asta conclusa vede il suo indirizzo memorizzato nello stato globale del Auction contract è dichiarato vincitore, e può procedere con la richiesta di trasferimento dell'asset secondo lo schema seguente:

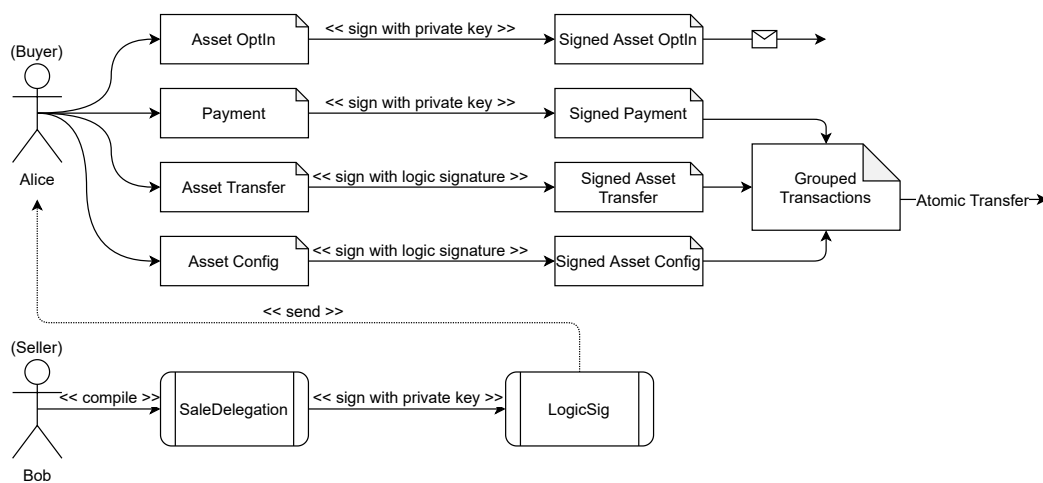


Figura 4.11: Riscossione dell'asset

Come si evince dalla Figura 4.11, il meccanismo di riscossione è implementato con lo stesso principio dell'acquisto diretto, salvo per la creazione dinamica della logicSig che in questo caso sarà creata appositamente per il trasferimento dell'asset oggetto dell'asta, e per l'utente che si è aggiudicato la sua proprietà.

Capitolo 5

Analisi e valutazione

In questo capitolo si analizzeranno alcune differenze tra la blockchain di Algorand e quella di Ethereum utilizzando come termine di paragone il caso d'uso implementato in sede di questo tirocinio.

Vedremo prima come le due blockchain implementano il modello di reward per i miners/minters (Sezione 5.1) e come questo influisca sui costi di gestione di una generica Dapp (Sezione 5.2) utilizzando come strumento di benchmark le applicazioni CryptoKitties ed AlgoKitties. Infine vedremo come si comportano i due sistemi in termini di prestazioni (Sezione 5.3) analizzando i tempi di conferma delle transazioni relative alle principali operazioni della game application.

5.1 Modello di ricompense

Qualunque sia il protocollo di consenso che una blockchain utilizza, questa prevede un sistema di ricompense a favore dei miners (o minters) che partecipano attivamente al protocollo. Il modo in cui queste ricompense vengono riconosciute agli utenti però, varia a seconda dell'algoritmo di consenso utilizzato, ed in particolare la differenza più sostanziale è quella che esiste tra gli algoritmi basati su PoW e quelli basati su PoS.

Mentre i primi richiedono una grande mole di lavoro derivante dalla complessità computazionale affrontata nella risoluzione del puzzle crittografico, i secondi non richiedono alcun calcolo complesso. Questo si traduce in un dispendio di energia molto maggiore per i modelli PoW rispetto ai modelli PoS e di conseguenza anche il modello di ricompense utilizzato nei due casi differisce. In particolare su Ethereum questo dispendio di energie è rappresen-

tato in termini di *gas*, un carburante virtuale che quantifica il lavoro svolto dal miner per l'approvazione di una transazione. Ogni unità di *gas* ha un valore pari ad una frazione di *Ether*, la valuta nativa di Ethereum, chiamata *Gwei* che costituisce il cosiddetto *Gas price* ($1\text{Gwei} = 0.000000001\text{Ether}$). La quantità di *gas*, moltiplicata per il prezzo unitario in *Gwei*, offerta al miner per l'approvazione di una transazione costituisce la sua ricompensa, detta *fee* (o commissione): maggiore è la ricompensa offerta (dal mittente), maggiore sarà il suo interesse nell'approvarla prima delle altre.

Tuttavia l'ammontare delle *fee* non dipende solamente dalla rapidità con cui richiediamo che la nostra transazione venga approvata, ma anche da quanto è complessa la transazione sottoposta. In generale le transazioni semplici (e.g. transazioni di pagamento) richiedono meno lavoro rispetto alle invocazioni di Smart Contracts, la cui complessità deriva direttamente dalla complessità della funzione invocata.

Su Algorand al contrario non vi è alcun calcolo complesso da effettuare, in quanto l'algoritmo di consenso PPoS utilizzato dalla blockchain richiede una quantità di lavoro fissa per ciascuna tipologia di transazioni che deriva dalla sola esecuzione della VRF. Di conseguenza anche il modello di ricompense utilizzato in Algorand è molto differente: l'utente paga una *fee* fissa (attualmente pari a 0.001Algo) che verrà depositata in un account chiamato *fee sink*. Ad intervalli di tempo regolari, vengono distribuite le *fee* raccolte dalle transazioni a tutti i nodi partecipanti attivamente al consenso. Il costo delle transazioni quindi dipende solamente dal valore corrente della cryptovaluta. Ad oggi non è stato chiarito come una eventuale crescita repentina di questo valore verrà affrontata al fine di mantenere le commissioni basse e costanti, ma vi è la proposta di istituire una *governance*¹ apposita, formata dai nodi della rete, che deciderà le politiche economiche nel lungo termine (i.e. abbassare le *fee* all'aumentare del prezzo della cryptovaluta).

Le analisi effettuate nel seguito del documento saranno basate sul prezzo medio che le due cryptovalute hanno mantenuto in un dato intervallo di tempo.

5.2 Analisi dei costi

Per poter stimare i costi di gestione nelle due diverse implementazioni bisogna prima di tutto definire quali sono i parametri ed i criteri utilizzati. Nell'applicazione originale, Cryp-

¹<https://algorand.foundation/faq#participation-rewards->

tokitties, le operazioni sono effettuate mediante invocazione degli Smart Contracts, mentre per quanto riguarda il porting sviluppato su Algorand gli stessi task sono implementati con l'approccio transaction-oriented descritto nella Sezione 4.2.

In riferimento a quanto detto nella sezione precedente, verranno considerati i valori registrati tra il 09/04/21 ed il 16/04/21. La tabella in Figura 5.1 mostra una descrizione dei vari task in termini delle funzioni invocate sull'implementazione Ethereum ed il relativo costo affrontato dall'utente per compiere ciascuna di esse, mentre la tabella in Figura 5.2 mostra i corrispondenti task sull'applicazione AlgoKitties in termini delle transazioni che li compongono.

Operazione	Funzione	Costo (Ether)	Costo (USD)
Trasferimento asset	<i>transfer()</i>	0.011996025	25.7252
Creazione asta	<i>createSaleAuction()</i>	0.018751320	40.2118
Piazzamento offerta	<i>bid()</i>	0.005660204	12.1382

Tabella 5.1: Stima dei costi per operazione (CryptoKitties)

Operazione	Transazioni	Costo (Algo)	Costo (USD)
Trasferimento asset	<i>Optin + Atomic Transfer</i>	0.004	0.0053
Creazione asta	<i>AppCall (create) + AppCall (updateCore) + AssetFreeze</i>	0.003	0.0040
Piazzamento offerta	<i>AppCall (Optin) + AppCall (placeOffer)</i>	0.002	0.0026

Tabella 5.2: Stima dei costi per operazione (Algokitties)

Le stime relative all'implementazione su Ethereum sono state ottenute calcolando una media su un set di esecuzioni dello stesso task, e utilizzando i valori estratti dal sito web *Etherscan.io*, che mostra per ciascuna transazione registrata sul ledger le informazioni relative al gas consumato, gas price corrente e commissioni totali, come mostrato in Figura 5.1.

Overview	Internal Txns	Logs (2)	State	Comments
Timestamp:	5 mins ago (Apr-12-2021 07:12:29 AM +UTC) Confirmed within 33 secs			
From:	0x1250cb1f4abb79a6cc1510536c525ba4c2d496e4			
Interacted With (To):	Contract 0xb1690c08e213a35ed9bab7b318de14420fb57d8c (CryptoKitties: Sales Auction) L TRANSFER 0.303619789496527779 Ether From CryptoKitties: Sales Au... To → 0xf7984f332a939a277b6d4c78... L TRANSFER 0.000018305905678617 Ether From CryptoKitties: Sales Au... To → 0x1250cb1f4abb79a6cc151053...			
Tokens Transferred:	From CryptoKitties: Sale... To 0x1250cb1f4abb7... For ERC-721 TokenID [1167647] CryptoKittie... (CK) 			
Value:	0.315467437850123062 Ether (\$689.50)			
Transaction Fee:	0.004095552 Ether (\$8.95)			
Gas Price:	0.000000083 Ether (83 Gwei)			
Gas Limit:	141,516			
Gas Used by Transaction:	49,344 (34.87%)			

Figura 5.1: Esempio di dati estratti da una transazione Ethereum

Per quanto riguarda il calcolo delle commissioni nell'implementazione Algorand i dati sono stati calcolati staticamente sulla base del prezzo della cryptovaluta e sulla quota fissa per transazione al momento dell'analisi. Notiamo come i costi in USD sono nettamente inferiori nell'implementazione Algorand dell'applicazione, e ciò è dovuto al prezzo più basso della cryptovaluta *Algo* rispetto ad *Ether*. Si potrebbe quindi dedurre che le due implementazioni abbiano costi di gestione analoghi a parità di valore della cryptovaluta, ma in realtà questo non è del tutto vero. Infatti, come già anticipato, Algorand mantiene dei costi di commissione bassi e punta ad adattarli all'andamento finanziario della cryptovaluta, con l'intento di mantenere i costi per gli utenti sempre entro un certo limite.

Ethereum al contrario segue un approccio di mercato libero, nel senso che a parità di gas consumato per il mining, il costo reale (i.e in USD/EUR) varia a seconda dell'inflazione e della volatilità della cryptovaluta.

5.3 Tempi di conferma

Prima di poter effettuare un'analisi sui tempi di conferma delle transazioni nelle due diverse implementazioni, è necessario definire un paio di concetti chiave:

- **Confirmation Time:** tempo che intercorre tra la sottomissione di una transazione e la sua approvazione, ovvero il suo inserimento nel ledger;

- **Transaction Finality:** indica il tempo dopo il quale la transazione precedentemente confermata può essere considerata inalterabile e quindi definitiva.

Su Ethereum la distinzione tra questi due concetti è fondamentale: quando una transazione viene approvata ed inserita nel ledger, non è detto che questa sia istantaneamente definitiva e inalterabile. Ricordiamo infatti, come descritto in Sezione 2.2.3, che la blockchain di Ethereum è soggetta al fenomeno delle fork. Ciò vuol dire che una transazione, pur essendo stata approvata, potrebbe finire in un ramo della blockchain che verrà successivamente scartato.

Per ovviare a tale problematica è opportuno attendere che un certo numero di blocchi, successivi a quello contenente la transazione considerata, vengano confermati e aggiunti allo stesso ramo del ledger. Si parla infatti di *Probabilistic Transaction Finality*, ovvero una stima per ridurre al minimo la probabilità che una transazioni finisca su una biforcazione non valida, e stabilire così quando una transazione possa considerarsi definitiva e appartenente alla blockchain.

Viceversa Algorand è una blockchain priva di biforcazioni, pertanto la transazione può immediatamente considerarsi definitiva dopo che è stata confermata, e i due valori sopra descritti, confirmation time e transaction finality, sono quindi coincidenti.

Le Figure 5.2, 5.3 e 5.4 mostrano l'output di esecuzione, rispettivamente, delle transazioni di trasferimento di un Kitten, creazione di un'asta e piazzamento di un'offerta.

```
Kitten menu:
  [1] Owned kittens
  [2] Kitten shop
  [3] Show kitten info
  [4] Buy kitten
  [0] Return to main menu
Choose an option: 4
Kitten ID: 80
  1) Optin Transaction...
Sent opt-in request, id: PFUDK7FLLIY3CSTHCOWQT3KQOS7NTIOSRTVUNX2GWYA4YHQDFRGA
Transaction PFUDK7FLLIY3CSTHCOWQT3KQOS7NTIOSRTVUNX2GWYA4YHQDFRGA confirmed in round 9487
  Time elapsed -> 5689.0 (ms)
  2) Making Atomic Transfer...
Transaction G7XGHHKHAQGFLS7ZS7FHUWSZ2V5HM53V5LTEB4NV4T0HZLRHKUAHQ confirmed in round 9489
  Time elapsed -> 7972.0 (ms)
Total task time: 13661.0 (ms)
```

Figura 5.2: Esempio di esecuzione del task "trasferimento" su AlgoKitties

```

Main menu:
  [1] Create kitten
  [2] Update core contract
  [3] Start auction
  [4] Close auction
  [0] Exit
Choose an option: 3
KittenID: 100
Starting price: 25000
Duration (rounds): 100
1) Freezing asset...
Transaction 4WYK7CR7XXDCUY3HUXZZX620BH4Y6SR5UKVVMFVP7PCREWQQM55Q confirmed in round 9909
Time elapsed -> 6251.0 (ms)
2) Deploying auction application...
Transaction 6SHICAWUFBDJFXZI72RJIVAV6PCT3SGMZLBVIA560EPT7RZZ24A confirmed in round 9911
Time elapsed -> 8059.0 (ms)
3) Updating coreApp...
Transaction I7L373RLTYXI6KSOHUGY7ALGYXWEAI5W64X3TRS3RKEJKPL7XP4A confirmed in round 9913
Time elapsed -> 7994.0 (ms)
Total task time: 22304.0 (ms)
-----

```

Figura 5.3: Esempio di esecuzione del task "creazione asta" su AlgoKitties

```

Auction menu:
  [1] Join auction
  [2] Place offer
  [3] Redeem kitten
  [4] Show auction status
  [0] Return to main menu
Choose an option: 2
How much do you want to offer? (micro Algos): 10000
1) Application Call (place offer)...
Transaction NST3UZXSIO5M6G4DIR2U55WDXZTJWSVCUZCLMJD3CIRWM7UPYVBA confirmed in round 9834
Time elapsed -> 5686.0 (ms)

```

Figura 5.4: Esempio di esecuzione del task "offerta asta" su AlgoKitties

La stima dei tempi medi di esecuzione dei task AlgoKitties, ovvero del tempo di approvazione delle transazioni che li compongono, è stata calcolata mediando i valori ottenuti da molteplici esecuzioni dello stesso task. Come già detto, infatti, queste transazioni sono da considerarsi immediatamente finalizzate e inalterabili, per cui non vi sono overhead e/o attese di alcun genere. I test sono stati condotti su rete privata, tuttavia il tempo medio di approvazione delle transazioni è analogo a quello stimato su reti pubbliche (testnet, mainnet)². In un futuro sviluppo di questo progetto, l'applicazione verrà deployata sulla Testnet

²<https://algoexplorer.io/>

per valutare quanto il congestionamento della rete influisca sui tempi d'approvazione delle transazioni.

Per il calcolo dei tempi di conferma dell'applicazione CryptoKitties sono invece stati simulati due diversi casi, in correlazione al problema delle biforcazioni precedentemente descritto:

- CASO 1: viene considerato solamente il *Confirmation Time*;
- CASO 2: viene aggiunto al *Confirmation Time* il tempo medio³ che intercorre tra la validazione di altri k blocchi successivi, con $k = 10$ e $k = 20$.

NOTA: I valori per k sono stati scelti in modo da garantire un buon livello di affidabilità senza compromettere eccessivamente i tempi di finalizzazione. Ciascun caso d'uso su Ethereum può definire un valore per questo parametro in modo arbitrario.

Sulla base dei criteri appena descritti sono stati ottenuti i valori riportati in tabella 5.3.

Operazione	Tempo di conferma (#blocchi confermati)			
	CryptoK. (0)	CryptoK. (10)	CryptoK. (20)	AlgoKitties
Trasferimento asset	0 min 37 sec	23 min 57 sec	47 min 17 sec	0 min 15 sec
Creazione asta	4 min 22 sec	27 min 43 sec	51 min 02 sec	0 min 24 sec
Piazzamento offerta	5 min 34 sec	28 min 54 sec	52 min 14 sec	0 min 06 sec

Tabella 5.3: Tempi di conferma dei task

È evidente come la natura della blockchain Ethereum renda estremamente dispendioso effettuare un determinato task, che invece può essere portato a termine su Algorand con un notevole risparmio economico e temporale. L'analisi del caso d'uso CryptoKitties mostra l'importanza dell'algoritmo di consenso utilizzato dalla blockchain, dimostrando l'enorme influenza che questo ha non solo sulle proprietà della blockchain stessa (scalabilità, decentralizzazione, sicurezza), ma anche su costi e prestazioni delle singole Dapp sviluppate su di essa.

³<https://etherscan.io/gastracker>

Capitolo 6

Conclusioni

Dagli studi condotti in sede di questo tirocinio sono emerse varie caratteristiche che rendono Algorand una blockchain molto promettente. Il suo algoritmo di consenso innovativo risolve le problematiche che affliggono le altre blockchain attualmente sul mercato, e rappresenta dunque un grande passo avanti per lo sviluppo di applicazioni decentralizzate. Sviluppi futuri della blockchain introdurranno nuove features e miglioreranno quelle esistenti, ampliando quelli che sono i possibili campi di applicazione.

Allo stato attuale della tecnologia si possono già immaginare molteplici contesti, nei quali l'impiego della blockchain Algorand comporterebbe significativi miglioramenti prestazionali e soprattutto un considerevole risparmio economico nel mantenimento delle stesse. In particolare il caso d'uso proposto, AlgoKitties, non è che un punto di partenza per cominciare a progettare delle valide alternative alle Dapps per la gestione e lo scambio di asset. Infatti, le tecniche proposte nell'implementazione della game application AlgoKitties, dimostrano come poter sfruttare le caratteristiche di questa blockchain per creare sistemi efficienti e al contempo sicuri.

Si pensi ad esempio a sistemi di gestione dei token di autenticazione, come il sistema *OAuth 2.0* ampiamente impiegato nei più diffusi social network come Facebook e Pinterest, del quale è stata recentemente proposta una implementazione basata sui token ERC-721 di Ethereum, da un gruppo di ricerca dell'università di Atene [22].

Un'ulteriore importante conseguenza scaturita dall'impiego dell'algoritmo PPoS è l'assenza di fork. Questo fattore è infatti determinante, ad esempio in applicazioni di finanza decentralizzata (*Deft*), in quanto riduce drasticamente i tempi di attesa previsti per la finalizzazione dei pagamenti.

In conclusione questo tirocinio può essere senz'altro considerato un una base per studi futuri in ambito blockchain, ed in particolare fornisce una visione generale di come poter sfruttare la tecnologia Algorand. Un prossimo passo potrebbe essere quello di portare su Algorand alcuni dei sistemi precedentemente sviluppati su rete Ethereum, come ad esempio i *Recommendation Systems*[23].

Inoltre potrebbe essere interessante sviluppare una versione più completa di AlgoKitties sfruttando i futuri aggiornamenti della blockchain, come ad esempio l'introduzione di Smart Contracts su layer-2 (ovvero off-chain) attualmente in fase di sviluppo. L'introduzione di questi oggetti, infatti, porterà su Algorand degli Smart Contract simili a quelli di Ethereum che utilizzano un linguaggio Turing-completo e permettono l'implementazione di funzionalità più complesse con un paradigma object oriented.

L'approccio transaction-oriented utilizzato in AlgoKitties, nonché l'utilizzo dell'Indexer per simulare un database distribuito di utenti e assets, ha permesso di ovviare alle attuali limitazioni del sistema ma ha inevitabilmente richiesto dei compromessi in termini di funzionalità ed efficienza. Pertanto una rivisitazione dell'applicazione AlgoKitties nel prossimo futuro consentirebbe sia di ampliare e migliorare le funzionalità attualmente presenti, sia di eseguire un confronto più approfondito con l'implementazione Ethereum CryptoKitties.

Riferimenti

- [1] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Rapp. tecn. Satoshi Nakamoto Institute, 2008.
- [2] www.ethereum.org. *A Next-Generation Smart Contract and Decentralized Application Platform*. URL: ethereum.org/en/whitepaper/#a-next-generation-smart-contract-and-decentralized-application-platform.
- [3] S. Micali, M. Rabin e S. Vadhan. «Verifiable random functions». In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. 1999, pp. 120–130. DOI: 10.1109/SFPCS.1999.814584.
- [4] Fabrizio Luccio Anna Bernasconi Paolo Ferragina. «Elementi di crittografia». In: Pisa University Press, 2015. ISBN: 978-88-6741-460-4.
- [5] Andrew Tar. *Proof-of-Work, Explained*. URL: cointelegraph.com/explained/proof-of-work-explained.
- [6] blockchain.com. *Bitcoin Hashrate Distribution*. URL: www.blockchain.com/pools.
- [7] Binance Academy. *Proof of Stake*. URL: academy.binance.com/it/articles/proof-of-stake-explained.
- [8] www.bitcoinwiki.org. *Delegated Proof of Stake (DPoS)*. URL: it.bitcoinwiki.org/wiki/DPoS.
- [9] EXBASE.IO. *exbase.io/en/wiki/bonded-proof-of-stake*.
- [10] www.medium.com. *The Proof-of-Stake Guidebook*. URL: medium.com/stakin/proof-of-stake-guide-dpos-vs-lpos-vs-bpos-vs-hybrid-1393a33e849c.

- [11] ethereum.org. *Representing arbitrary data as an Eth-based asset*. URL: ethereum.org/en/nft/.
- [12] Dong Ku David Im. «The technology trade-offs among the security, decentralization, and scalability of blockchain». In: *The Blockchain Trilemma*. 2018, pp. 14–20.
- [13] Algorand Foundation. *Algorand's Pure PoS approach*. URL: www.algorand.com/what-we-do/technology/pure-proof-of-stake.
- [14] Algorand Foundation. *Algorand Ecosystem Overview*. URL: www.algorand.com/ecosystem#Technical.
- [15] Algorand Foundation. *Transaction Execution Approval Language (TEAL)*.
- [16] Algorand Foundation. *The Algorand Smart Contract Language*. URL: developer.algorand.org/docs/features/asc1/teal/.
- [17] Algorand Foundation. *Stateful Smart Contracts Overview*. URL: developer.algorand.org/docs/features/asc1/stateful/.
- [18] Algorand Foundation. *Algorand Atomic Transfers*. URL: developer.algorand.org/docs/features/atomictransfers/.
- [19] Algorand Foundation. *Indexer V2*. URL: developer.algorand.org/docs/features/indexer/.
- [20] Dieter Shirley, Mack Flavelle e Benny Giang. *CryptoKitties: Collectible and Breedable Cats Empowered by Blockchain Technology*. Rapp. tecn. Dapper Labs.
- [21] Jason Paulos. e Shumo Chu. *PyTeal: Algorand Smart Contracts in Python*. URL: pyteal.readthedocs.io/en/latest/overview.html.
- [22] V. A. Siris et al. «OAuth 2.0 meets Blockchain for Authorization in Constrained IoT Environments». In: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. 2019, pp. 364–367. DOI: 10.1109/WF-IoT.2019.8767223.
- [23] Andrea Lisi et al. «Rewarding reviews with tokens: An Ethereum-based approach». In: *Future Generation Computer Systems* 120 (2021), pp. 36–54. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2021.02.003>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X21000480>.

- [24] Silvio Micali. «ALGORAND: The Efficient and Democratic Ledger». In: *CoRR* abs/1607.01341 (2016). arXiv: 1607.01341. URL: <http://arxiv.org/abs/1607.01341>.
- [25] Yossi Gilad et al. «Algorand: Scaling Byzantine Agreements for Cryptocurrencies». In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. Shanghai, China: Association for Computing Machinery, 2017, pp. 51–68. ISBN: 9781450350853. DOI: 10.1145/3132747.3132757. URL: <https://doi.org/10.1145/3132747.3132757>.