POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Domain Trusted Transaction

L'utilizzo di una Blockchain *permissionless* quale strumento di certificazione di informazioni

Relatore
prof. Maurizio Rebaudengo

Candidato
Lorenzo Manicone

Consoft S.p.A
dott.ssa Serena Ambrosini

Anno accademico 2019-2020

A mamma e papà.

Sommario

Punto di partenza di questa *tesi* è l'esperienza di tirocinio svolta presso la sede torinese di Consoft S.p.A. L'argomento centrale della dissertazione in questione verte attorno alla tecnologia della *Blockchain Ethereum* collocata nell'ambito del progetto *PININ*, con particolare enfasi agli aspetti riguardanti la *certificazione dell'identità digitale*. Il lavoro è stato suddiviso genericamente in tre fasi. La prima, prettamente *teorica*, ha visto un periodo di formazione e ricerca sulle tecnologie della *Blockchain*. In seguito alla definizione dei requisiti e del tipo di applicativo richiesto, è stata portata avanti una seconda fase di sviluppo, implementazione e testing dello stesso. In una terza fase è stata invece redatta tale dissertazione. La fase di formazione teorica è stata svolta per la maggior parte presso la sede aziendale di *via Pio VII, 127 - 10127 Torino*, mentre le successive fasi sono state concluse in remoto, a causa delle restrizioni imposte dalla pandemia di *Covid-19*.

Consoft S.p.A.

Consoft Sistemi S.p.A. [2] è un'azienda italiana di ICT fondata nel 1986, oggi facente parte del Gruppo Consoft. In particolare, Consoft Sistemi offre una vasta gamma di Servizi e Soluzioni in ambito ICT. Realizza progetti *end to end* attraverso attività di consulenza tecnologica e metodologica, soluzioni integrate e servizi in insourcing/outsourcing. Ha sedi a Torino, Milano, Genova, Roma e Tunisi, 400 dipendenti e un fatturato superiore a 27 milioni di Euro. Accanto alla capogruppo Consoft Sistemi sono attive altre 2 società: CS InIT, specializzata nello scouting e distribuzione di soluzioni software e Consoft Sistemi MEA per espandere l'offerta della capogruppo nel mercato nord-africano e medio-orientale.

Progetto PININ

Il progetto PIemuNt chèINa (PININ)[3] mira a incrementare tanto la qualità, quanto la percezione della stessa, dei prodotti agro-alimentari Piemontesi di fascia alta, innestando tecnologie per tracciabilità e autenticazione, in modo da introdurre innovazione nella commercializzazione di prodotti della filiera alimentare e garantire la protezione dei diritti di proprietà intellettuale dei marchi agro-alimentari Piemontesi.

Le tecnologie innovative portanti del progetto, Blockchain, Intelligenza Artificiale e Big Data, IoT, Realtà Aumentata, ma anche il Geoweb e i Processi di Business, permetteranno contemporaneamente di creare un sistema innovativo di

tracciatura dei prodotti alimentari lungo tutta la filiera, dalle materie prime al consumatore, e di introdurre servizi innovativi per il consumatore. L'obiettivo è anche quello di evitare gli sprechi in un'ottica di economia circolare, facilitando la gestione dei prodotti in scadenza, promuovendo i prodotti a KM0 e introducendo controlli nella catena alimentare per certificare la sostenibilità tramite piattaforme tecnologiche.

I sistemi di raccolta automatizzata dei dati di tracciabilità attualmente adottati dalle aziende agroalimentari sono stati per lo più concepiti per la tracciabilità interna delle singole aziende, e risultano scarsamente condivisi e disomogenei sia relativamente alla tecnologia utilizzata che alla tipologia del dato trattato, per cui si verifica il moltiplicarsi di diverse applicazioni (app, siti web, database privati e istituzionali) che penalizzano l'efficienza di filiera, la precisione del dato tracciato e l'esperienza del singolo utente costretto ad utilizzare diverse app o siti e copiare il codice del lotto del prodotto. Infine la centralizzazione del sistema di tracciabilità su singole aziende mina la fiducia del consumatore.

Il progetto PININ mira, pertanto, a costruire un'infrastruttura distribuita e decentralizzata basata su Blockchain, che permetta una tracciabilità a livello di lotto e che sia scalabile lungo tutta la filiera. La stessa tecnologia Blockchain permetterà anche la tracciabilità dell'utilizzo dei fondi Europei per l'allevamento per quel che riguarda il bestiame nei pascoli alpini.

Obiettivo finale è quello di realizzare una piattaforma in ottica Regionale che fornisca servizi senza favorire o tutelare una singola azienda/filiera, ma in prospettiva possa diventareuna ricchezza partecipata da tutti gli attori coinvolti

Tale progetto, che vede Consoft S.p.A. come capofila, prevede la collaborazione di numerose altre aziende. I temi centrali sono molteplici, come la gestione della sicurezza e della privacy, della interoperabilità, della governance di processi e servizi e in particolare dell'**identità digitale**. Ed è proprio il modulo dell'identità digitale che costituisce il cuore della corrente dissertazione.

Scopo della tesi

Rispetto al progetto PININ, la corrente dissertazione si colloca nel contesto della tecnologia Blockchain Ethereum. Nel dettaglio, si è cercata una soluzione che permettesse ai vari soggeti in gioco da un lato di vedere certificate le proprie informazioni e dall'altro di verificare l'autenticità e la veridicità di informazioni pubblicate da terzi. Punto focale dell'applicativo sviluppato è dunque una possibile strategia di certificazione delle informazioni mediante l'utilizzo di tecnologie basate sui principi della Blockchain.

Ringraziamenti

Un sentito ringraziamento ad Alberto Ferrini e a Luca Demma, per il loro supporto costante, sia teorico che pratico, indispensabile nella realizzazione dei capitoli della mia tesi.

Ringrazio l'azienda Consoft S.p.A. per avermi dato la possibilità di svolgere il mio lavoro di tesi in un ambiente innovativo e dinamico.

Ringrazio il mio relatore, il prof. Maurizio Rebaudengo, per la sua disponibilità e per la fiducia riposta nel mio progetto.

Ringrazio infine le persone che mi sono state vicine lungo il mio percorso universitario. Un sentito grazie a tutta la mia famiglia, alle mie sorelle, ai miei genitori, ai miei nonni, alle mie zie e ai mie zii, ai miei cugini e alle mie cugine. Ringrazio infine tutti i miei amici e colleghi, senza i quali il mio percorso non sarebbe stato lo stesso.

Indice

Capitolo 1

Introduzione

Negli ultimi anni, l'interesse nei confronti delle tecnologie riguardanti la *Blockchain* ha subito un notevole impulso. Si tratta di tecnologie dotate di un ampio ventaglio applicativo, che non riguarda solamente la pura *computer science*, ma anche e soprattutto contesti economici e industriali. Inizialmente legata al Bitcoin[1] e al mondo delle criptovalute, la *Blockchain* si è poi dunque allargata a diversi ambiti applicativi, rivelando la sua utilità per molteplici industrie.

1.1 Una moneta *digitale*

Fin dagli albori del web, durante gli anni '90, sulla falsariga degli ideali di libertà su cui la rete internet si stava fondando, nasceva l'idea di avere una moneta *digitale*. Una tecnologia di questo tipo si sarebbe basata su strumenti crittografici e matematici che ne avrebbero garantito autenticità, integrità e privacy. Un potere virtuale di questo tipo avrebbe dunque permesso delle libertà alle quali sarebbe stato impossibile accedere mediante un economia reale e *centralizzata*.

Nel 1983 furono inventate da un celebre docente di crittografia, David Chaum, le *blind signatures*. Si trattava di uno strumento crittografico capace di inviare e firmare un valore numerico identificabile e modificabile senza essere rivelato. Ciò diede vita alle moneta *chaumiane*. Tuttavia si trattava di una soluzione ancora centralizzata, che avrebbe avuto necessità dell'approvazione di una banca statale per poter essere sviluppata appieno. Nel corso degli anni Novanta nascevano dunque molteplici compagnie impegnate a distribuire e gestire monete digitali. David Chaum diede infatti vita alla Digicash nel 1990, mentre nel 1996 nasceva negli USA la E-Gold, nel 1998 e nel 1999 nascevano rispettivamente Benz e Floor, anche loro basate su una tecnologia fondamentalmente centralizzata. La stessa celeberrima Paypal, in una fase iniziale, ambiva alla creazione di una moneta globale, virando poi sulla gestione dei pagamenti elettronici.

Si tratta di tecnologie, come detto, basate su monete *centralizzate*, prive di un loro valore intrinseco e impossibilitate a procedere se prive di finanziamenti esterni.

Un primo importante passo verso la moderna Blockchain si verificò nel 2004, quando Hal Finney inventò la *Reusable Proofs of Work*. La **RPOW** è un sistema

centralizzato di pagamento la cui moneta era basata su ciò che viene chiamato **proof-of-work**. La moneta digitale implementata da Finney restò tuttavia poco più che un esperimento, fino al diffondersi delle teorie di Satoshi Nakamoto[4].

1.2 Satoshi Nakamoto

Nel 2008 **Satoshi Nakamoto** riuscì a garantire ad una moneta la caratteristica di decentralizzazione. Ciò veniva fatto mediante una rete *peer-to-peer*, una struttura dati condivisa chiamata **Blockchain** e un insieme di regole, che daranno vita ai concetti di **Consenso** [4]. Fino alla pubblicazione del paper di Nakamoto, non era possibile garantire ad una rete peer-to-peer aperta una tolleranza consistente al **guasto bizantino**, dovuto al *Problema dei Generali Bizantini*. Questo tipo di guasto prevede che un gruppo di nodi malevoli, fra loro coordinati, possa sempre attaccare il sistema con successo.

1.3 Il problema dei generali Bizantini

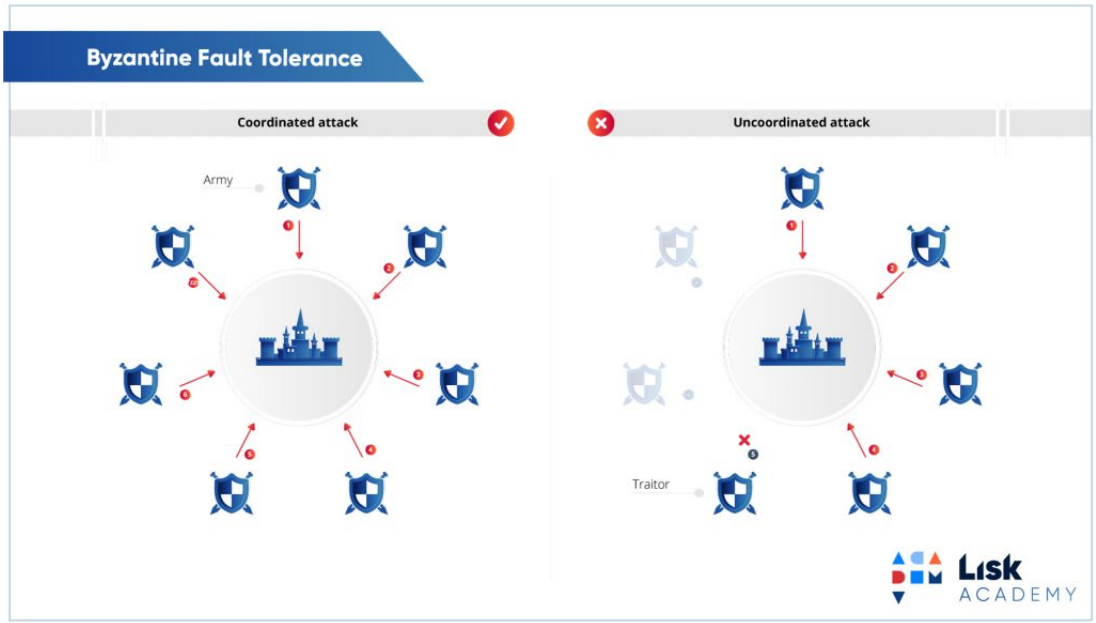

Figura 1.1: Il problema dei generali Bizantini

Si tratta di un *problema informatico* in cui più parti cercano di trovare un accordo mediante lo scambio di messaggi. Il problema fu ideato da Leslie Lamport nel 1982 [6]. Il Problema dei Generali Bizantini vede un gruppo di generali bizantini impegnati a gestire un assedio ad una città. In particolare, devono decidere se ritirarsi o attaccare. Le loro truppe si trovano schierate ad una certa distanze le une dalle altre all'interno della città stessa. Ogni generale potrebbe dunque manifestare diverse preferenze, addirittura quella del tradimento. I generali possono comunicare solamente mediante dei messaggeri, al fine di raggiungere una decisione unanime.

Si noti che un attacco o una ritirata eseguita da una sola parte delle truppe genererebbe un fallimento peggiore di un attacco coordinato di una ritirata coordinata. Nel corso degli anni, sono stati inventati una serie di algoritmi finalizzati a trovare un *consenso* in reti di questo tipo, senza tuttavia ottenere soluzioni del tutto efficienti ed efficaci. Tale problema sarà risolto dalla soluzione proposta da Nakamoto, basata sulla teoria dei giochi e sulle tecniche di **PoW**. Viene introdotto dunque il protocollo **Bitcoin**, una soluzione innovativa al problema dei guasti bizantini.

Nasceva dunque una nuova moneta digitale, una *criptomoneta*, il **Bitcoin**, che garantiva transazioni sicure ma soprattutto era priva di un'entità centrale che la gestisse. A differenza delle normali monete, infatti, non esistono entità simili alle banche. Dunque, una fondamentale caratteristica del Bitcoin è la sua decentralizzazione e indipendenza da una qualsiasi Trusted Third Part.

Satoshi Nakamoto è lo pseudonimo dell'inventore della **criptovaluta Bitcoin**. Attualmente non si è a conoscenza se si tratti di una singola persona o di un gruppo. Nonostante molti personaggi del panorama *scientifico-informatico* siano stati sospettati di nascondersi dietro Satoshi Nakamoto, nessuno di essi ha mai rivendicato la paternità dei lavori in questione. In ogni caso, in seguito alla pubblicazione del suddetto paper, questi diede vita alla *prima* **Blockchain**.

Capitolo 2

La Blockchain

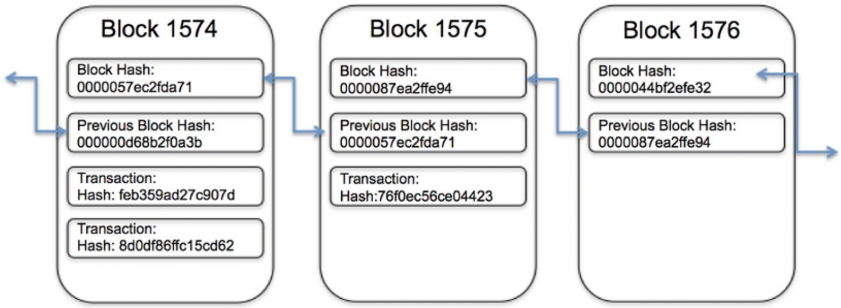

Figura 2.1: Una catena di blocchi

La **Blockchain** è una struttura dati condivisa e immutabile, rappresentata come un *merkle tree*. Si tratta di un registro digitale distribuito, un *distribuited ledger*, capace di mantenere dati in modo sicuro, verificabile e permanente attraverso strumenti di crittografia. I dati in questione vengono chiamati *transazioni*, le quali vengono raggruppate in blocchi. Tale struttura dati è composta dunque da una *catena di blocchi*, in continua crescita, collegati fra loro mediante un algoritmo di *hash*. In particolare, ogni blocco contiene l'*hash* del blocco precedente, garantendo alla Blockchain la sua caratteristica di immutabilità. Ogni nodo possiede una copia dell'intera *Blockchain*, non esistendo alcuna copia centralizzata ufficiale di riferimento, garantendo una parità fra i vari nodi. Si tratta dunque di un *database* massivamente distribuito. I dati non possono infatti essere modificati in maniera retroattiva, in quanto cambiamenti nei blocchi passati genererebbero blocchi futuri differenti. In tal caso, sarebbe necessario il consenso del 51% della rete per *approvare* una modifica di questo tipo. Dunque la Blockchain, pur contemplando la modifica dei dati, può essere considerata *secure by design*, fornendo un sistema di computazione distribuita con un'elevata tolleranza al problema dei generali bizantini.

Tabella 2.1: Header blocco

Versione
Hash del blocco precedente (PrevHash)
Merkle root
Timestamp
Bits
Nonce
Numero di transazione

2.1 Blocchi e transazioni

2.1.1 Struttura

Un blocco è composto da un *body* e da un *header*. Il body contiene un numero variabile di *transazioni*, che varia in base alla loro dimensione, anch'essa variabile, organizzate in un *Merkle Tree*. L'header contiene invece sette campi utili per la gestione del blocco stesso. Inoltre, il blocco contiente un timestamp UNIX che specifica il momento di validazione del blocco. Infine è importante menzionare il *nonce*. Il campo *nonce* viene settato dal *miner* in modo che l'hash del blocco calcolato sia minore o uguale al target attuale della rete, ossia la sua *difficoltà*. I miner calcolano il valore del nonce diverse volte, finchè non ne ottengono uno adeguato.

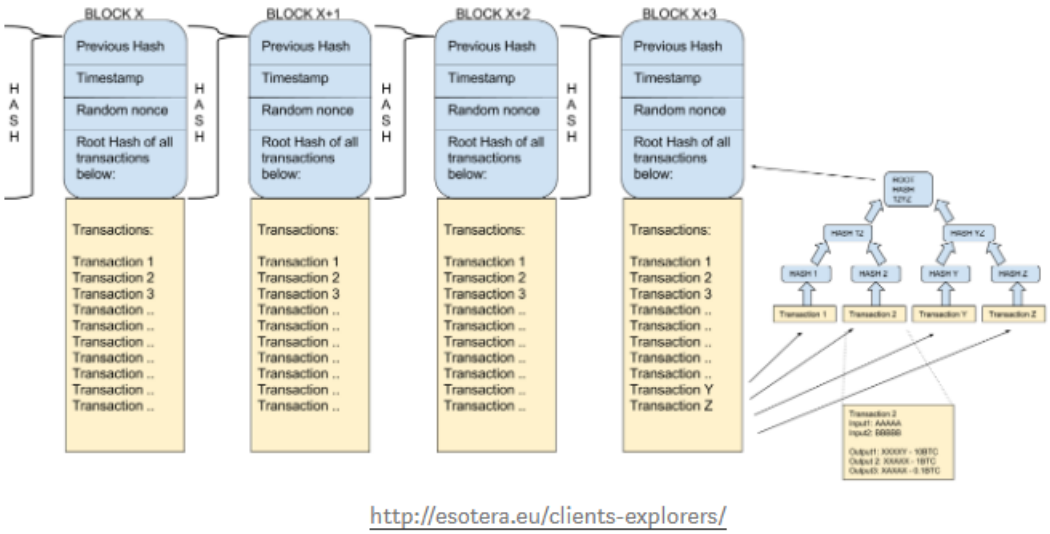

Figura 2.2: Struttura dei blocchi.

2.1.2 Validazione dei blocchi e *mining*

Il processo di validazione e registrazione di nuovi blocchi sul ledger della Blockchain è detto **mining**. Nel caso della Blockchain Bitcoin si basa sull'algoritmo

di Proof-of-Work [5] e avviene ogni 10 minuti, mentre in *altre Blockchain* i tempi possono essere differenti. I blocchi vengono *minati* cioè caricati sulla blockchain, da entità dette *miner*. Ogni miner sceglie un insieme di transazioni da minare in base al compenso che ogni transazione conferisce. Inoltre, per ogni blocco minato, i miner ricevono una *fee*, ossia un ulteriore compenso. Si noti che si tratta, ovviamente, di remunerazioni in termini di criptovalute. Un miner, dunque, seleziona le transazioni generate ancora non validate *suggerendo* alla rete un potenziale nuovo blocco. L'insieme delle transazioni non ancora validate è detto *transaction pool*. A questo punto viene applicata un funzione crittografica di hash al blocco stesso, in modo che l'output rispetti dei precisi parametri. Il primo miner che riesce a *risolvere* il blocco lo trasmette in rete, andando di fatto ad *allungare* la catena. In questo contesto è possibile che vi siano biforcazioni della Blockchain, dette *fork* che andranno poi di fatto *tagliate* via, dando vita a blocchi orfani. Infatti, è possibile che blocchi multipli vengano risolti contemporaneamente. Convenzionalmente, viene scartata la catena più corta, mentre quella più lunga viene considerata *fidata* dalla maggioranza dei miner. Il processo di mining è molto costoso in termini di risorse computazionali e spesso viene eseguito da gruppi di macchine progettate *ad-hoc* per scopi di questo tipo. In particolare, la *difficoltà* del mining aumenta con l'aumentare della gente che vi partecipa. Si parla di **mining pool**, dunque un insieme di utenti che mettono in comune la propria potenza computazionale in modo da eseguire *mining* più velocemente ed ottenere un guadagno, successivamente suddiviso su base contributiva. La crescita in termini di dimensioni di tali mining pool rischia di minare il principio di decentralizzazione proprio della Blockchain, essendo oggi le maggiori di esse a disposizione di una potenza di calcolo tale da raggiungere il 51% del totale. Ciò significa essere in grado di *effettuare un attacco alla rete*.

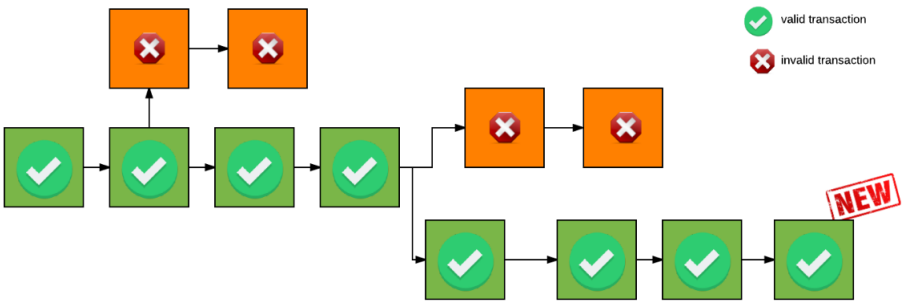

Figura 2.3: Esempio di *fork*.

Il *consenso* viene dunque raggiunto mediante il concetto di Proof of Work, strettamente legato al problema dei Generali Bizantini. I miner per poter caricare un blocco devono risolvere tale PoW, di difficoltà variabile. Gli altri utenti verificheranno la riuscita di tale operazione, e solo in seguito a tale controllo il blocco verrà effettivamente caricato in Blockchain.

L'algoritmo di **PoW** ha un consumo molto elevato in termini di corrente, stimato nell'ordine dei *TWh*.

Si noti che il PoW non è l'unico algoritmo utilizzabile in questo contesto, ma è sicuramente il più diffuso. Una valida alernativa è rappresentata dall'algoritmo di

Proof-of-Stake. Sommariamente, PoS si basa sull'idea di fornire potere di mining in modo proporzionale alla quantità di moneta posseduta.

2.2 Il ruolo della Crittografia

Nel mondo della Blockchain viene data grande importanza ai sistemi di **crittografia asimmetrica**, con particolare riferimento alla coppia chiave pubblica-privata. Ogni utente è *identificato* dalla propria chiave pubblica, che ne costituisce l'indirizzo. Un utente può accedere al proprio *fund* solamente mediante la propria chiave privata, mentre una transazione è invece valida solo se *firmata* dal mittente.

2.2.1 Crittografia a chiave pubblica o *asimmetrica*

Se il concetto di *chiave simmetrica*, anche non in termini prettamente informatici, è estremamente antico, quello di **chiave pubblica** è un concetto del tutto *moderno*, nato negli anni '70.

L'idea di base è quella di utilizzare una coppia di chiavi, pubblica e privata, con funzionalità *reciproche*. Ogni utente genera due chiavi, tenendone una *privata e una pubblica*. In tal modo, una chiave, una qualsiasi delle due, viene utilizzata per cifrare e l'altra per decifrare. Ciò che viene cifrato con una chiave può tuttavia essere decifrato *solamente* con l'altra chiave. Ogni chiave può dunque essere utilizzata sia per cifrare che per decifrare. In ogni caso, gli algoritmi asimmetrici sono molto lenti e costosi, caratterizzati da un alto carico di elaborazione e in genere si utilizzano a supporto di algoritmi simmetrici. In particolare, vengono utilizzati per diffondere le chiavi simmetriche e *non* si utilizzano per criptare grandi moli di dato. E', infatti, impensabile cifrare dati di grandi dimensioni con tecniche *asimmetriche*. I principali algoritmi sono Diffie-Hellman, RSA[7], DSA, El Gam.

2.2.2 Firma digitale

Un importante campo di applicazione dei sistemi di crittografia asimmetrica è quello della **firma digitale**. La firma digitale è un concetto molto vicino alla cifratura asimmetrica dei dati con la chiave privata dell'autore, ma si noti che non sono la stessa cosa. Solitamente non si cifrano direttamente i dati ma un loro riassunto, detto *digest*. L'utilizzo di una firma digitale fornisce **autenticazione e integrità dei dati**.

Ovviamente, è necessario conoscere la chiave pubblica del mittente per poter verificare l'integrità e autenticità dei dati in ingresso. Riguardo la tramissione di un generico messaggio M:

- (firmatario S) $H = enc(S_{K_{pri}}, hash(M))$. L'hash di M viene cifrato con la chiave privata.
- (trasmissione) $M||H$. Messaggio M e hash cifrato H vengono concatenati e trasmessi.

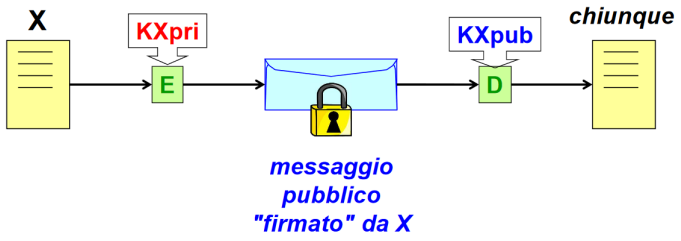

Figura 2.4: *Firma digitale.*
Solo X può generare tali bit, e dunque si ha la certezza matematica che il messaggio sia stato scritto da X. Un utente cifra un messaggio con la propria chiave privata. Per poterlo decifrare, occorrerà la chiave pubblica del mittente. Dunque chiunque sarà in grado di decifrare quell'oggetto, ma sarà stato *firmato* da *quel* mittente. Dunque si ha la certezza *matematica* che quell'oggetto sia stato scritto da un *preciso* mittente. Questo è il *principio* su cui si andrà a basare la *firma digitale*. Precisamente, nella firma digitale è un *digest*, riassunto, che viene cifrato.

- (verificatore V) $X = dec(S_K pub, H)$. H viene decifrato con la chiave pubblica del mittente e ottenendo X.
- (verifica) *if (X == hash(M)) then OK else ALARM!* Si verifica che X sia uguale all'hash di M.

Il *digest* di generico messaggio corrisponde allo stesso fatto passare attraverso una generica funzione di *hash*.

2.2.3 Le funzioni di hash

Il *digest* è dunque un "riassunto" a **lunghezza fissa** del messaggio da proteggere, che può avere qualsiasi lunghezza, e deve cercare di mantenere il più possibile le caratteristiche del massaggio originale. Il digest in genere si calcola mediante opportune *funzioni di hash* crittografiche. E' importante che una funzione di hash sia *deterministica*, ossia che dato un input l'output non cambi, e sia computazionalmente veloce, a prescindere dall'input.

Funzioni di hash dedicate

Non tutte le tecniche di hashing sono adatte ad applicazioni del genere. Ad esempio, strategie come CRC, usato in ethernet, non sono adatte alla crittografia. CRC è utilizzato per rilevare errori quali interferenze elettromagnetiche, e dunque per errori casuali. Funzioni di questo tipo non vanno bene per applicazioni crittografiche. Solitamente, le funzioni di hash dedicate:

- Dividono il messaggio M in N blocchi $M_1 ... M_N$.
- Applicano ripetutamente una funzione base (f).
- $V_k = f(V_{k-1}, M_k)$ (cioè risultato precente, blocco corrente da cifrare) con $V_0 = IV$ e $h = V_N$.

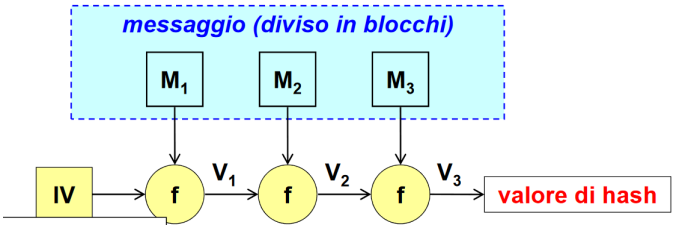

Figura 2.5: *Funzioni di hash dedicate.*
Nelle funzioni di hash dedicate, ogni blocco viene combinato con il precedente. L'output di tali funzioni ha una lunghezza fissa. Il vettore di inizializzazione non è critico, è fisso e dipende dall'algoritmo. Inoltre, non necessita di segretezza.

Gli algoritmi di hash si distinguono in base alla dimensione del blocco e alla lunghezza del digest. Alcuni esempi sono MD2, MD4, MD5, SHA-1, (famiglia) SHA-2, SHA-3.

La **lunghezza del digest** è importante per evitare *aliasing*, cioè collisioni. Gli algoritmi si differenziano in base alla lunghezza del digest e alla dimensione del blocco. Algoritmi come MD2, MD4, MD5 sono, ad oggi, *obsoleti*. Se l'algoritmo è ben ideato e genera un digest di N bit, allora la probabilità di aliasing è $P_A = 1/2^{Nbit}$.

In particolare, nelle reti **Blockchain**, le funzioni di hashing vengono utilizzate nell'algoritmo di PoW per calcolare i blocchi validi ricercando in nonce che soddisfi la corrente difficoltà della rete. Bitcoin, in particolare, utilizza SHA-256[8].

2.3 Trustless, trasparenza e scalabilità

L'innovazione portate dalla tecnologia Blockchain ha introdotto un nuovo concetto fondamentale, quello della *trustless*, letteralmente, *assenza di fiducia*. Nei più popolari e tradizionali sistemi centralizzati la cosiddetta *fiducia* veniva riposta in un ente centrale esterno, come una banca o un generico fornitore di pagamenti, il quale si poneva come garante di generiche transazioni. Il sistema tradizionale è dunque un sistema centralizzato in un ente che si occupa solo di garantire le transazioni[9]. Nel caso della Blockchain, tale ente viene sostituito dalla Blockchain stessa. Difatti, la fiducia non viene del tutto *eliminata*. Tuttavia, la fiducia che viene riposta sulla Blockchain si fonda su un principio di decentralizzazione che non intende del tutto eliminare il principio di fiducia, ma *distribuirlo* fra i miner, i quali si occupano di garantire e confermare le transazioni. I miner, dal canto loro, sono spinti a cooperare in virtù del ritorno economico che otterrebbero. La *trustless* della Blockchain è pertanto una fiducia nella crittografia e nella matematica, in particoalre nella crittografia a chiave pubblica e nel meccanismo del consenso. Si parla dunque di **fiducia distribuita**.

Venendo inoltre meno la presenza di un ente centrale, viene meno il potere che questi poteva esercitare sulla trasanzione stessa. Ad esempio, in seguito ad un versamento, un generico ente avrebbe potuto procedere con l'investimento finanziario

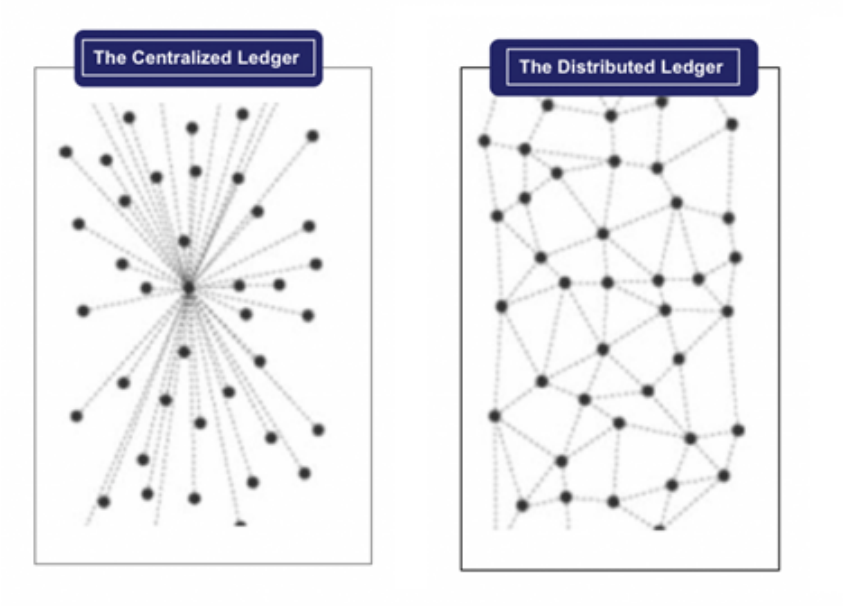

Figura 2.6: *Ledger* centralizzato o distribuito

.

del deposito effettuato. Si parla dunque di *trasparenza*. La partecipazione di una terza parte, ruolo giocato dalla rete stessa nel caso della Blockchain, viene richiesta solo nella conferma delle transazioni, mentre la gestione del proprio portafoglio è del tutto nelle mani del singolo utente. Precisamente con il termine *trasparenza*, si indica la condivisione online di tutte le informazioni. In particolare, è possibile analizzare e visualizzare liberamente la cronologia della transazioni, notando che i proprietari della criptovaluta vedono comunque la propria riservatezza protetta.

Un altro importante concetto da analizzare è quello di *scalabilità*, ossia il numero di transazioni gestibili al secondo. I sistemi basati su Blockchain, in questo caso, sono molto indietro rispetto a quelli tradizionali. Se Paypal gestisce 193 transazioni al secondo, VISA 24000, Bitcoin ne riesce a gestire soltanto 7 [9]. Tale ritardo è dovuto ai tempi necessari per inserire una transazione e raggiungere il consenso. In ogni caso, esistono soluzioni a questo problema, come l'aumento della dimensione del blocco o l'utilizzo di Proof-of-Stake.

In ogni caso la combinazione di fattori quali la fiducia distribuita, la trasparenza dell'operato, la condivisione delle informazioni e il continuo innovarsi sull'efficienza della scalabilità rendono il sistema della Blockchain un'alternativa estremamente valida al sistema tradizionale.

Privacy

In realtà la Blockchain non è del tutto esente dal problema della privacy. Infatti, conoscendo l' indirizzo-chiave pubblica di un utente, è possibile risalire alla cronologia delle sue transazioni. Un altro problema può scaturire dalla tracciabilità degli indirizzi IP nel momento in cui si eseguono transazioni. Conviene dunque usare il proxy TOR, che permette di rendere anonimo il traffico di rete.

2.4 Blockchain *permissionless* o *permissioned*

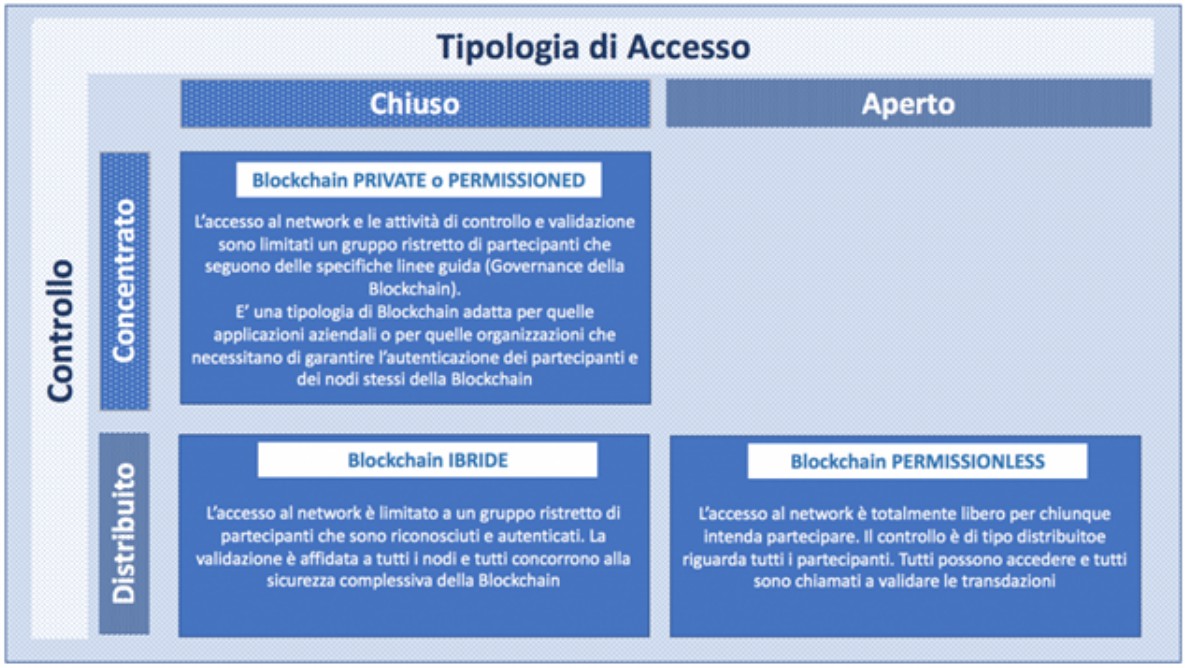

Figura 2.7: Blockchain *permissionless* o *permissioned*.

Una Blockchain può essere definita pubblica, *permissionless*, o privata, *permissioned*. Una Blockchain pubblica, ad esempio Bitcoin, non prevede alcun tipo di autorizzazione per l'accesso alla rete, per eseguire transazioni o partecipare alla verifica di nuovi blocchi. Viene garantito l'anonimato, sono del tutto *open source* e nessun utente ha privilegi rispetto agli altri o alla Blockchain stessa. Principale problema delle Blockchain permissionless è quello della scalabilità.

Una Blockchain privata è invece soggetta ad un'autorità centrale che ne gestisce gli accessi. Si tratta dunque di un gruppo chiuso. Ogni utente può avere ruoli differenti, con privilegi differenti. Nelle reti permissioned viene introdotto il concetto di governance e centralizzazione in un sistema concepito come decentralizzato e distribuito. Dunque può avveinre una gestione dell'identità, con l'identitificazione dei partecipanti. La presenza di un minor numero di nodi aumenta l'efficienza del *consenso* e dunque la scalabilità della rete. Un rete permissioned può essere sia *open source* che sviluppata privatamente.

2.5 Bitcoin

Il Bitcoin, concepito da Nakamoto nel 2008, fu la prima implementazione della tecnologia della Blockchain. Nel dettaglio, si tratta di una **criptovaluta** decentralizzata che può essere scambiata liberamente fra gli utenti sulla rete peer-to-peer Bitcoin.

Nel protocollo Bitcoin la quantità di criptomoneta *minabile* è limitata a 21 milioni. Infatti, il compenso del mining di un blocco viene dimezzato ogni 210000

Figura 2.8: Bitcoin utilizza PoW come algoritmo di consenso supportato dalla funzione di hash SHA-256 e ha un *block time* di 600 secondi.

blocchi minati, e considerando che un blocco viene minato ogni 10 minuti, ciò accade ogni 4 anni, circa. In questo contesto si colloca il concetto di **difficoltà**. Considerando che il mining dipende sia dall'*hashrate* che dalla potenza di calcolo messa a disposizione da sistemi informatici in continua evoluzione, occorre che la cadenza del mining di un singolo blocco resti costante. Il *block time* viene mantenuto stabile dalla rete modificando il parametro di *difficoltà*. Più aumenta la potenza di calcolo a disposione della rete, più aumenta la difficoltà, e viceversa. In particolare, la *difficoltà* rappresenta il numero di *zeri* con cui deve terminare l'hash del blocco calcolato.

Bitcoin ed economia

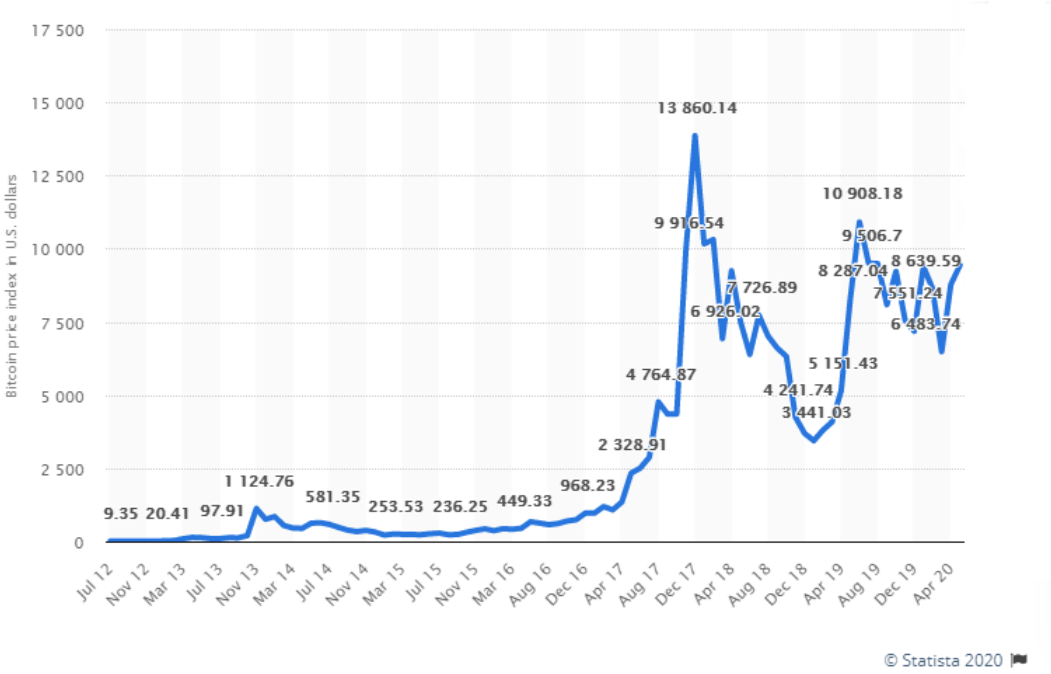

Figura 2.9: Evoluzione del valore di bitcoin dal Gennaio 2012 al Maggio 2020[11] , in dollari USA.

Per comprendere appieno l'impatto che Bitcoin ha avuto sulla finanza, si consideri l'evoluzione che tale criptovaluta ha avuto in termini di forza economica. Il tasso di cambio di partenza, nel 2009 stabiliva un valore di 1 USD pari a 1309 BTC [10]. Il 17 dicembre del 2017, circa dieci anni dopo, un bitcoin valeva più di 13000

dollari americani per poi avere un rapido crollo [11]. Si tratta, in ogni caso, di un tasso di cambio estremamente fluttuante, volatile e spesso instabile.

A livello di notazione, ci si riferisce alla moneta con il termine *bitcoin*, con la "b" iniziale minuscola, mentre si utilizza la "b" maiuscola per riferirsi alla tecnologia in sè, *Bitcoin*.

2.6 Vulnerabilità della Blockchain

2.6.1 Double spending attack

La Blockchain, ovviamente, non è un sistema infallibile, essendo anch'essa vulnerabile ad un serie di attacchi, un esempio dei quali può essere appunto il *double spending attack*. Il *double spending attack* consiste nell'inviare nella Blockchain due transazioni in rapida successione in modo da spendere due volte la stessa moneta. Tali transazioni si trovano contemporaneamente nel transaction pool, in attesa di essere confermate. Nel caso in cui una transazione venga verificata da un miner prima dell'altra, la secondo verrà automaticamente scartata. Tuttavia, se le due transazioni vengono scelte simultaneamente da due distinti miner si andrà a creare una biforcazione della catena, che verrà poi risolta scartando la catena più corta. Una transazione viene infatti ritenuta valida dopo un minimo di 6 conferme, ossia il numero di blocchi minati dopo il blocco contenente la transazione.

2.6.2 Il rischio del *51% attack*

Nel caso in cui un'entità, che sia un miner o un gruppo, divenga in grado di detenere il controllo del 51% dell'*hashpower* totale, potrebbe essere in grado di estendere la catena a partire da un blocco già pubblicato, creando di fatto una catena di blocchi parallela che cresca più rapidamente di quanto lo faccia la sequenza di blocchi originale. Tale fenomeno, detto *chain reorganization*, permetterebbe di eseguire operazioni di *double spending*, di annullare o modificare le *conferme* di transazioni passate, di prevenire *conferme* di nuove transazioni o addirittura di evitare che altri miner riescano a minare nuovi blocchi.

In ogni caso non si tratterebbe di un potere *assoluto*. Non sarebbe difatti possibile annullare, modificare o prevenire le transazioni di altri, generare moneta a proprio piacimento o effettuare transazioni con moneta appartenente a terzi.

2.7 Zero Knowledge Proof

La **Zero-knowledge Proof** o ZKP, letteralmente "dimostrazione a conoscenza zero" è uno schema di crittografia proposto dai ricercatori del MIT Silvio Micali, Shafi Goldwasser e Charles Rackoff negli anni '80. Con una strategia di questo tipo, una parte, detta *prover*, è in grado di dimostrare che una specifica dichiarazione sia vera per l'altra parte, detta *verificatore*, senza rivelare ulteriori informazioni.

Dunque la ZKP permette ad un'entità di dimostrare essere a conoscenza di una certa informazione senza rivelare l'informazione stessa. Si tratta di un algoritmo crittografico che richiede comunque tempi relativamente lunghi e che risulta essere imperfetto, in quanto non è inesistente la possibilità che un soggetto malevolo sia in grado di convincere un verificatore di un'affermazione falsa. Ciò è dovuto alla natura probabilistica della ZKP.

Una ZKP può essere di tipo interattivo o non interattivo.

Dal punto di vista della *blockchain*, le transazioni ZKP sono utili per mantenere un alto livello di anonimato e dunque di privacy. In particolare, tale sistema permette di provare che un utente abbia a disposizione moneta sufficiente per trasmettere una transazione senza rivelare chi sia o quanta moneta abbia.

2.8 Proof of Stake

Il **Proof-of-Stake**[12] è un protocollo utilizzato per gestire la sicurezza di una rete Blockchain richiedendo agli utenti di dimostrare il *possesso* di una certa somma di criptovaluta, differenziandosi dalla classica *PoW* non utilizzando algoritmi di *hashing* per validare transazioni elettroniche. Il *PoS* viene dunque utilizzato per raggiungere un consenso distribuito, risolvendo il problema dell'eccessivo consumo di corrente elettrica tipico del *PoW*.

In particolare, il *consenso* viene raggiunto richiedendo agli utenti di bloccare una certa somma di valuta in modo da avere la possibilità di essere selezionati per validare un blocco di transazioni e ricevere la relativa remunerazione. Tale somma congelata viene considerata una *caparra* per la validazione del blocco, una vera e propria "messa in gioco"[1]. Dunque, più un utente tiene bloccata una somma, maggiore è la possibilità che questi venga selezionato come *validatore* del blocco successivo.

Proof-of-Stake, oltre a risolvere il problema dell'efficienza energetica, aggiunge protezione ad attacchi di tipo *51%attack* e garantisce maggiore decentralizzazione non essendo possibile la creazione di mining pool, tipici dei contesti di applicazione del *PoW*.

Nel dettaglio, quando un blocco viene aggiunto alla *chain*, è necessario che venga scelto il creatore del blocco seguente. Questi non può chiaramente essere colui il quale possiede più criptovaluta, altrimenti sarebbe sempre lo stesso. Esistono più metodi di selezione:

- *Selezione casuale.* Si utilizza un funzione di hash che prende in considerazione la somma messa in gioco dai vari utenti, il cui risultato è facilmente prevedibile.
- *Selezione sulla base dell'anzianità.*

[1]La locuzione *Proof-of-Stake* è traducibile in italiano come "prova che si ha un interesse in gioco"

- *Selezione sulla base della velocità.*
- *Selezione sulla base del voto.* In una soluzione di questo tipo, creatori del blocco vengono scelti sia sulla base della posta in gioco, appunto lo *stake*, sia mediante una selezione tramite votazione. Il voto della comunità incentiva i creatori dei blocchi ad agire in maniera responsabile, ma apre a scenari del tipo *sybil attack*, in cui un utente possa impersonare più di un delegato.

E' bene sottolineare che il protocollo di *Proof-of-Stake* sta prendendo piede in maniera consistente nel mondo delle criptovalute, tanto che Ethereum, la seconda moneta per capitalizzazione, è in una fase di transizione da un sistema PoW a uno PoS.

Capitolo 3

Ethereum

Figura 3.1: Logo di Ethereum.

Nel corso del tempo si sono sviluppate nuove e differenziate reti Blockchain, delle quali Bitcoin era solamente la prima. In particolare si fa riferimento alla rete **Ethereum** [14], che vide la luce nel 2015.

3.1 Cenni storici

Se l'identità dell'inventore della rete Bitcoin è pressochè incerta, la paternità della rete *Ethereum* è ben definita e appartiene a Vitalik Buterin, un ricercatore e programmatore russo. Questi, fin da giovane, si interessò alle neonate tecnologie della *catena di blocchi*, cofondando, nel 2011, all'età di 17 anni, la rivista *Bitcoin magazine*. Dopo aver compreso il potenziale di una tecnologia di questo genere, ne analizzò le limitazioni. Infatti il linguaggio di scripting di Bitcoin poteva essere utilizzato solo per determinati tipi di transazioni, mentre una tecnologia di questo tipo, se correttamente evoluta, avrebbe potuto risolvere problemi computazionli ben più complicati.

Buterin propone dunque la rete **Ethereum**. Non si tratta solamente un network per lo scambio di valore monetario, ma di una rete su cui far girare dei veri e propri programmi, detti *contratti*. La prima menzione della piattaforma avvenne nel 2013 [13] ad opera del russo sulla sua stessa rivista, mentre la sua prima implentazione fu rilasciata il 30 luglio 2015.

3.1.1 Analisi economica

Se in una fase iniziale il cambio USD-ETH rimaneva stabile sul rapporto 1:10, dal 2017 il valore del ether ha subito impennate notevoli, toccando un picco di circa 1000 dollari americani nel gennaio 2018 [17].

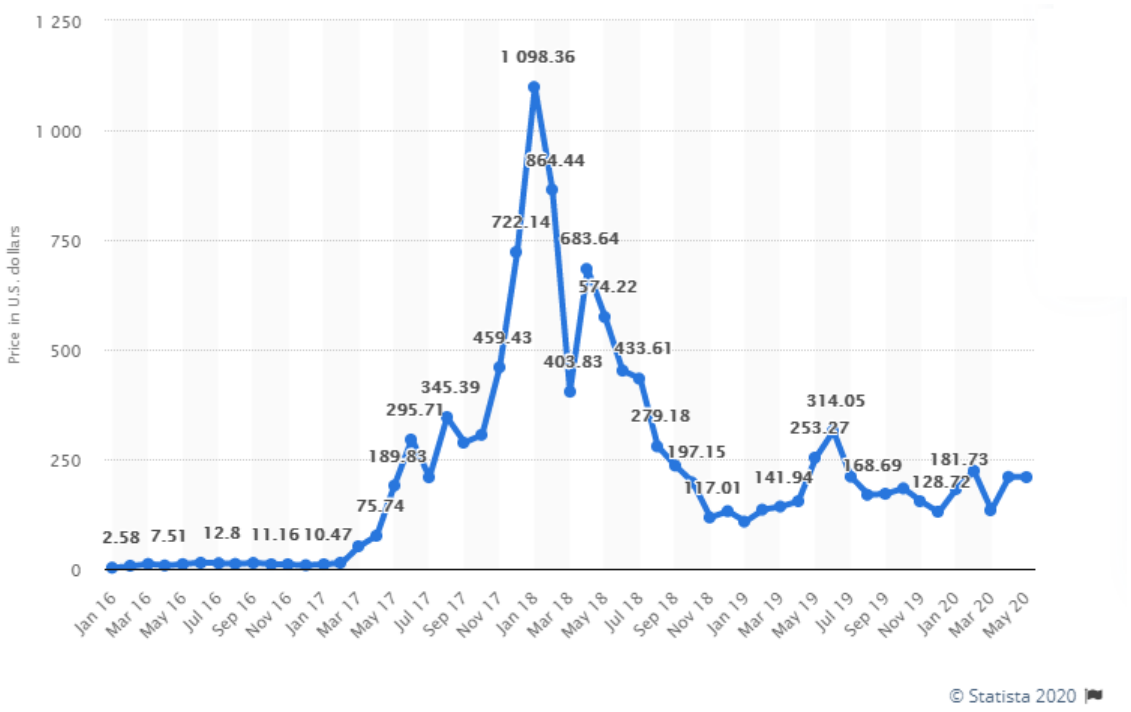

Figura 3.2: Evoluzione del valore di ether fra il Gennaio 2016 e il Maggio 2020 [17], in dollari USA.

3.1.2 Il Trilemma della scalabilità

Il *trilemma della scalabilità* [15] è stato coniato dall'ideatore di Ethereum Vitalik Buterin. Un *trilemma* è un problema a tre variabili di cui solo due possono essere soddisfatte contemporaneamente. Nel caso della blockchain le variabili in questione sono la sicurezza, il decentramento e la scalabilità, intesa come il numero di transazioni che possono essere processate nell'unità di tempo. Si consideri che una Blockchain non potrà mai essere più prestante di un database tradizionale, *centralizzato* per antonomasia, ed occorre dunque trovare un giusto compromesso fra decentralizzazione e velocità. Inoltre, un sistema sufficientemente robusto viene garantito dall'algoritmo di PoW, che tuttavia svantaggia la velocità. Altri algoritmi, quali *Proof of Stake, Delegated Proof of Stake, Proof of Authority*, prediligono fattori di velocità a discapito di sicurezza e decentramento. In ogni caso è impossibile parlare di soluzione perfetta, in quanto ognuna dipende dal contesto in cui viene applicata.

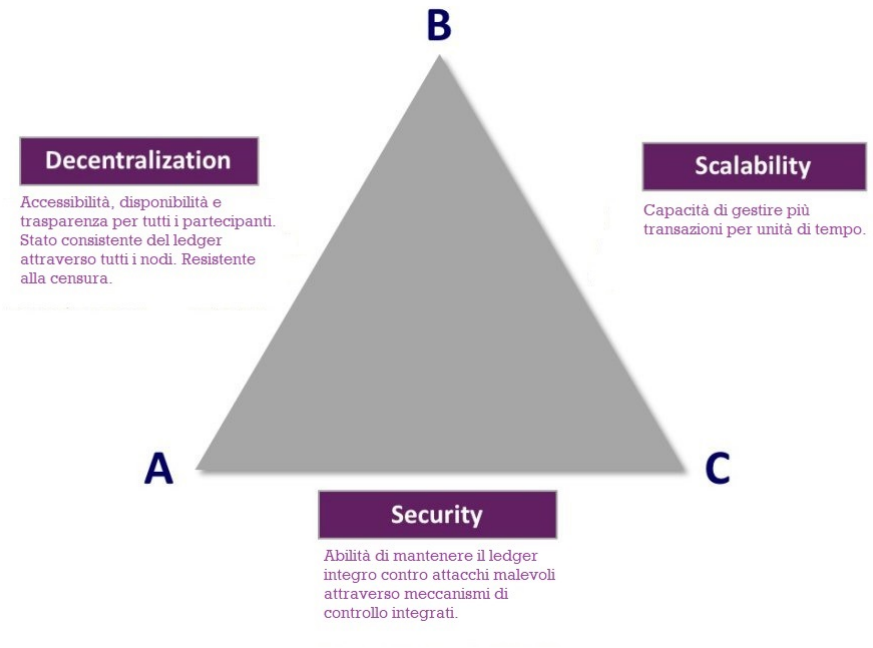

Figura 3.3: Il Trilemma della scalabilità.

3.2 Una Blockchain *programmabile*

Ethereum [16] è dunque una Blockchain *programmabile* e la sua criptovaluta nativa è chiamata **ether**, ETH. Ha molte caratteristiche simili al Bitcoin, potendo essere inviata a chiunque, ovunque e in qualsiasi momento senza essere controllata da alcun ente centrale. Ethereum aggiunge la possibilità di essere appunto *programmata*, e dunque gli sviluppatori possono usarla per creare applicazioni decentralizzate, dette **DApp**, *decentralized application*. Si tratta di applicazioni estremamente attendibili, in quanto, una volta caricate sulla rete, queste restano immutabili. Ethereum mette a disposione la *ethereum virtual machine* (EVM) che permette di eseguire script con l'ausilio di un network di nodi pubblici.

Ethereum è dunque una comunità legata alla Blockchain, estremamente diffusa e attiva, è open-source ed è portata avanti da una moltitudine di persone, provenienti da tutto il mondo.

Per il PoW, Ethereum usa un algoritmo chiamato *Ethash*, che riduce il vantaggio del mining attraverso l'utilizzo di dispositivi ASIC. Un *nodo Ethereum* è un dispositivo generico che utilizza il protocollo Ethereum. I nodi non hanno identità a lungo termine e sono liberi di generare un numero qualsiasi di coppie di chiavi asimmetriche.

Le tipologie di applicazioni sviluppabili sulla piattaforma di Ethereum sono svariate, quali portafogli per criptovalute[1], applicazioni finanziarie o mercati decentralizzati. Inoltre la possibilità di eseguire script sulla EVM, una **macchina virtuale**

[1]consentono di effettuare pagamenti istantanei ed economici con ETH o altri beni

globale decentralizzata, permette l'implementazione di casi d'uso quali salvataggio e notarizzazione dei dati, finanza decentralizzata e soprattutto **gestione e certificazione dell'identità digitale**.

Tabella 3.1: Differenze fra Ethereum e Bitcoin

Ethereum	**Bitcoin**
Linguaggio turing completo	Linguaggio non turing completo
I dati sono organizzati in due strutture dati dissociate. Dati aggiuntivi e saldi non sono mantenuti nella blockchain, bensì su un Merkel Patricia Tree.	Struttura dati unica
Costo transazione dipende da complessità computazionale, uso di banda e memoria utilizzata.	Costo transazione dipende solo dai byte della transazione.
Block time di 15 secondi	Block time di 10 minuti
Mining produce ether in modo consistente	Compenso dimezzato ogni 4 anni
Ether in rete non è limitato	Bitcoin limitati a 21 milioni

3.2.1 Ethereum Virtual Machine

La **Ethereum Virtual Machine (EVM)**, è dunque la macchina virtuale che permette l'uso degli script all'interno della blockchain di Ethereum. Tali programmi, detti **Smart Contract**, sono scritti in un linguaggi di programmazione di alto livello, **Solidity**. Uno *Smart Contract* è un account autonomo controllato via software, mentre è detto *wallet* un account umano[2]. Una **transazione** è un messaggio mandato da un account ad un altro e può includere dati binari o ether.

La EVM è dunque l'ambiente di *runtime* per i suddetti *Smart Contract*. Completamente isolata dal sistema host, non può accedere alla rete, al filesystem o a processi esterni. Possiede uno *stack* di 1024 elementi, con due diverse tipologie di storage, temporaneo, *memory*, che vede la memoria eliminata alla conclusione della transazione, o permanente, *storage*, mantenuto permanentemente in Blockchain. Nello *stack* avvengono tutte le computazioni.

Per evitare implementazioni che potessero portare a problemi di creazioni del consenso, si è scelto di ridurre in maniera consistente l'insieme di istruzioni disponibili. E' possibile sviluppare operazioni aritmetiche sui bit, comparazioni ed espressioni logiche. E' consentito fare salti condizionati e incondizionati, con un totale di 130 *opcodes*.

[2]Ethereum prevede due tipi di account, external, controllati da coppie chiave pubblica-privata, o contracts, controllati dal codice memorizzato con l'account.

3.2.2 GAS

Lo sviluppo di un linguaggio *turing-completo*, seppur limitato rispetto ai più celebri linguaggi di programmazione, ha introdotto una serie di istruzioni relativamente *potenti*, fra le quali la possibilità di effettuare *cicli*. nasce dunque il problema della terminazione. Si introduce il concetto di Gas, che costituisce il *costo* di una transazione. Ogni transazione ha un *gas-limit*.

Le transazioni di Ethereum hanno, dunque, un *costo*. Ognuna viene infatti pagata in ciò che viene chiamato **GAS**. Il *GAS* rappresenta una quantità di ether proporzionata all'operazione richiesta dalla transazione, e ha lo scopo di limitare l'ammontare del massimo carico di esecuzione della transazione stessa.

Nel momento in cui la EVM esegue una transazione, viene dunque consumato GAS sulla base di specifiche regole. Può verificarsi che il GAS a disposizione di una transazione termini prima della conclusione della transazione stessa. In tal caso si verifica una `out-of-gas exception`, che provoca un *rollback* di tutte le modifiche apportate dalla transazione.

Dunque il meccanismo del GAS assolve a due principali problemi. In primo luogo un'esecuzione non può girare oltre una certa quantità prepagata, in secondo luogo un validatore vede la garanzia di ricevere tale somma prepagata anche in caso di fallimento dell'esecuzione.

3.2.3 Smart Contract

Come si è detto, la blockchain Ethereum è una blockchain *versatile*, programmabile dagli utenti e si differenzia dalla blockchain di Bitcoin poiché possiede un linguaggio Turing completo, *Solidity*. In Ethereum una transazione può costituire non solo un trasferimento di ether da un utente ad un altro, ma anche il deploy o il richiamo di uno **Smart Contract**[3].

Uno **Smart Contract** è un programma, che talvolta può rappresentare il ciclo di vita di un contratto, le cui condizioni sono codificate ed applicate via software. Si tratta di codice eseguito su blockchain. Gli Smart Contract possono *interagire* fra loro e si contraddistinguono per una serie di caratteristiche:

- Hanno un ID univoco in blockchain.
- Sono dotati di un saldo di criptovaluta.
- Possiedono una *memoria privata* e del *codice eseguibile*.

Si definisce **stato di un contratto** la coppia composta dal *saldo del contratto* e dalla sua memoria memoria. Lo *stato* viene salvato in blockchain e aggiornato ogni volta che il contratto viene *invocato*. Ad ogni contratto salvato in blockchain viene

[3]si noti che nel corso della dissertazione si farà riferimento al concetto di Smart Contract, per semplicità, con il termine Contratto.

assegnato un indirizzo univoco di 20 byte[4]. Una volta che il contratto è pubblicato in blockchain, il codice eseguibile non è più modificabile.

Infine, per eseguire uno Smart Contract, occorre mandare una transazione al suo indirizzo. Il contratto viene dunque eseguito dai nodi miner della rete in modo da raggiungere il consenso sull'output della chiamata. Infine, lo stato del Contratto viene aggiornato. Sulla base delle transazioni che riceve, un Contratto può leggere o scrivere nella sua memoria privata, trasferire criptovaluta da e verso il suo saldo, o semplicemente scambiare messaggi con altri utenti o contratti, o addirittura creare nuovi contratti. Si noti che il codice dei contratti Ethereum è scritto in bytecode.

3.2.4 DApp

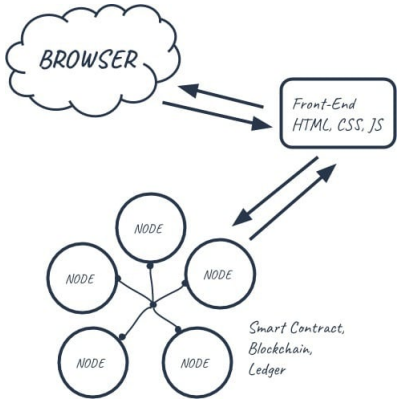

Figura 3.4: DApp

Le DApp, *Decentralized Application*, sono delle applicazioni che si interfacciano con contratti presenti sulla rete Ethereum in modo da unire i vantaggi delle applicazioni tradizionali con le potenzialità degli Smart Contract. Si tratta generalmente di applicazioni web, finalizzate a fornire un'interfaccia grafica e maggiore interattività all'utente. Sia lato client che lato server si appoggiano ad un gruppo di Smart Contract per gestire l'interazione con la Blockchain.

3.2.5 Solidity

Gli Smart Contract, dunque, non sono altro che *programmi*. I linguaggi di programmazioni con cui tali Contratti possono essere sviluppati sono molteplici, in particolare *Serpent* basato su Python, *LLL* basato su Lisp e *Solidity*, basato su Javascript. Fra questi, il più popolare è di certo **Solidity** [18]. Si tratta di un linguaggio di programmazione in continua evoluzione, *objected-oriented*, sviluppato per essere simile alla sintassi *ECMAScript* [19], in modo da risultare maggiormente *familiare* agli sviluppatori web.

[4]gli indirizzi sono composti dal prefisso "0x" che ne indica la base esadecimale e rappresenta i 20 bytes più a destra dell'hash Keccak-256 della chiave pubblica ECDSA.

Solidity supporta l'*ereditarietà*, sia singola che multipla, permette l'importazione di librerie e la creazione *tipi di dato* definiti dall'utente. Si tratta dunque di un linguaggio *tipizzato*. I Contratti scritti con Solidity sono corredati con un Application Binary Interface, **ABI**. L'ABI permette allo Smart Contract di interfacciarsi, eventualmente, con altre applicazioni.

Un contratto scritto in Solidity è simile ad una *classe* tipica dei linguaggi objected-oriented. Uno Smart Contract può dunque contenere dichiarazioni di variabili di stato, funzioni, modificatori di funzioni, strutture ed *eventi*[5] o può anche consistere in una libreria o in un'interfaccia.

Tipi di variabili

Solidity è dunque un linguaggio *tipizzato* [20], prevedendo infatti vari tipi di variabili. Una variabile *di stato* viene salvata permanentemente nello *storage* del Contratto.

- Variabile di tipo *valore* - viene passata *by value*. Si tratta di variabili booleane, interi, numeri a virgola fissa, *indirizzi*, esadecimali o funzioni.
- Variabile di tipo *riferimento* - può essere modificata mediante differenti *riferimenti*. Occorre specificare in quale area di memoria è stata salvata, dunque in `memory, storage` o `calldata`. Se in `memory`, la sua esistenza è limitata alla chiamata della funzione, se in `storage` la variabile viene collocata nell'area di memoria permanente in qualità di variabile di stato. Infine, `calldata` è la porzione di memoria che contiene gli argomenti delle funzioni chiamate da funzioni esterne, dunque da *altri* Smart Contract.
- Variabile di tipo *mapping* - rappresenta una struttura dati di tipo *chiave-valore*. I mapping vengono salvati nella memoria *storage*.

In aggiunta ai tipi di dato primitivo, quali `uint, int` e `address`, il linguaggio Solidity supporta i tipi di dato `array` e `string`. In realtà, quest'ultimo è implementato internamente come un `array`. Solidity permette, mediante l'utilizzo di `struct`, la creazione di tipi di dato utente. Si noti infine che in Solidity non esiste il concetto di *null* e le variabili dichiarate possiedono sempre un valore di default che dipende dal tipo.

Funzioni

Le funzioni rappresentano il codice eseguibile di uno Smart Contract. Le funzioni che non modificano lo stato della Blockchain sono etichettate come `view`, mentre le funzioni `pure` non possiedono l'accesso neppure in lettura allo stato del contratto. Le funzioni `payable` possono invece ricevere *pagamenti*. Una funzione `payable`

[5] il concetto di evento verrà esposto nei paragrafi seguenti

può essere chiamata solamente mediante una transazione che viene registrata nella Blockchain e il chiamante può allegare un `value` in ETH alla transazione stessa.

Una funzione di un contratto può richiamare funzioni di un altro contratto pubblicato ad un altro indirizzo, purchè ne conosca l'ABI, e ovviamente l'indirizzo.

Le funzioni possono avere vari tipi di *visibilità*:

- `public`. Funzioni dichiarate come `public` fanno parte dell'interfaccia del contratto e possono essere chiamate sia internamente che esternamente.
- `external`. Una funzione `external` fa parte dell'interfaccia del contratto, può essere chiamata da contratti esterni ma non può essere chiamata internamente.
- `internal`. Un funzione `internal` è accessibile solo internamente.
- `private`. Le funzioni `private` sono visibili solo dal contratto nel quale sono state definite.

Eventi

Gli eventi rappresentano un'astrazione della funzionalità di log della EVM. Applicazioni possono mettersi in ascolto di un evento, attendendone l'emissione, tramite RPC o tramite un generico client Ethereum. Una volta *emesso*, un evento vede i propri argomenti salvati nel *transaction log*[6].

Dunque gli eventi, a cui si fa riferimento mediante la parola chiave `event`, possono essere usati per salvare dati in Blockchain a basso costo, in quanto richiedono un consumo di GAS ridotto rispetto alle variabili `storage`.

Gestione degli errori

Solidity gestisce gli errori mediante un sistema di *eccezioni*. Nel caso in cui venga lanciata un'eccezione, lo stato precedente del Contratto viene ripristinato e viene mandato un messaggio di errore al chiamante della funzione. Le eccezioni si propagano automaticamente nello stack di chiamate.

Esistono delle funzioni per la gestione delle eccezioni, quali `assert`, utilizzata per testare errori interni al codice, o `require`, utilizzata per controllare parametri di input e delle pre-condizioni richieste da una funzione. Inoltre, un'eccezione lanciata da `assert` consuma tutto il GAS a disposizione della chiamata.

3.2.6 Sicurezza

In ogni caso, gli Smart Contract possiedono diversi punti deboli, che possono essere talvolta utilizzati da eventuali attaccanti. Occorre dunque gestire in fase di

[6]si tratta di una struttura dati interna alla Blockchain inaccessibile ai contratti

programmazione al meglio la sicurezza del codice, in modo da renderlo robusto in termini di *security*.

Ad esempio, vanno gestiti i possibili *underflow* e *overflow*, poichè la EVM definisce tipi dati di dimensione *fissa*. Esiste inoltre il problema della **re-entrancy**. Fra le funzionalità offerte dagli Smart Contract sviluppati su Ethereum persiste la possibilità di invocare ed utilizzare codice di Contratti esterni, inviando eventualmente ether in maniera autonoma. Chiamate esterne possono dunque essere usate per *forzare* l'esecuzione di codice aggiuntivo mediante funzioni `fallback`. Tali funzioni possono essere implementate da un attaccante scatendando l'esecuzione di codice malevolo nel contratto attaccato. Esistono varie strategie per sopperire a questa falla di sicurezza. E' possibile, ad esempio, bloccare la modifica delle variabili di stato prima che una funzione di un altro smart contract venga invocata, rimuovendo il blocco una volta terminata l'esecuzione.

3.2.7 Reti di Test

Rispetto alla rete Ethereum, esistono delle implementazioni della stessa il cui unico scopo sia quello di fornire agli sviluppatori una rete su cui testare le proprie applicazioni, prima di pubblicarle, eventualmente, sulla rete Ethereum vera e propria. Si tratta di reti che tentano di assomigliare il più possibile alla rete Ethereum, fornendo ether in maniera gratuita, rispettando tuttavia alcune regole di *antispam* Le reti di test possono essere sia pubbliche che private. Se la rete private più popolare è di certo Ganache, le reti di test pubbliche degne di nota sono perlomeno Rinkeby e Ropsten.

Capitolo 4

Identità digitale

Quello della definizione del concetto di *identità digitale* è uno dei problemi più antichi del web. Se nel mondo fisico esistono degli standard per verificare l'identità, quali l'utlizzo di passaporti, carte di identità o patenti di guida, ciò non è vero nel mondo di internet, in cui non esiste uno standard univocamente riconosciuto che verifichi e definisca l'**identità digitale**.

4.1 La necessità di uno standard

Se nel mondo fisico la verifica delle certificazioni di identità viene effettuata da attori *umani*, nel mondo digitale risulta necessario utilizzare un formato standard, in quanto il controllo viene eseguito da *macchine*. In secondo luogo occorre uno standard per verificare la fonte e l'integrità delle credenziali digitali (mediante l'utilizzo di una firma digitale, per cui è necessaria una modalità di verifica). Sono dunque necessari sistemi a crittografia asimmetrica.

Ad oggi, lo standard più diffuso rispetto alla rappresentazione di credenziali digitali è quello della **W3C**, il World Wide Web consortium[21].

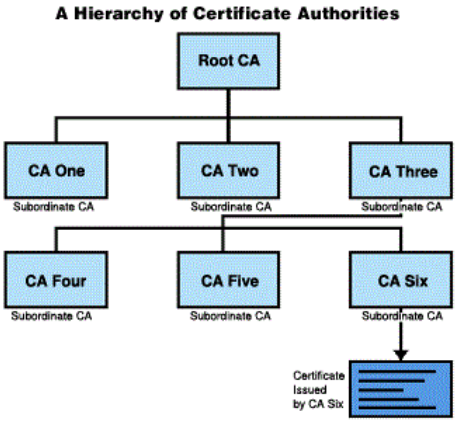

Figura 4.1: La struttura gerarchica delle Certification Authority.

Inoltre, per eseguire questo tipo di verifica si fa affidamento sulle **PKI**, *public key cryptography infrastructure*. Tale sistema si basa sulla premessa secondo cui chiunque possieda la chiave pubblica può verificarne la relativa firma digitale.

In questo sistema è fondamentale il ruolo delle **CA**. Le Certification Authority rilasciano certificati garantendo l'identità di un certo utente, organizzandosi secondo una struttura gerarchica. Un errore da parte della CA risulta tuttavia, fatale.

L'utlizzo delle PKI comporta, in ogni caso, un serie di problematiche, quali la *centralizzazione*, il loro essere *ingrombranti* e *costose* e una generale lentezza nei processi di ottenimento di revoca di certificati da parte delle CA.

4.2 SPID, la soluzione della PA

In Italia, la soluzione adottata dalla Pubblica Amministrazione per l'identità digitale è il sistema **SPID** [22], Sistema Pubblico di Identità Digitale. In particolare, SPID permette ai cittadini di accedere ai servizi online delle Pubbliche Amministrazioni e dei soggetti privati con un'unica Identità Digitale.

SPID è basato sul framework SAML - Security Assertion Markup Language - sviluppato e manutenuto dal Security Services Technical Committee di OASIS [23], che permette la realizzazione di un sistema sicuro di Single Sign-On (SSO) federato. Grazie a SAML, un utente può accedere ad una moltitudine di servizi appartenenti a domini differenti effettuando un solo accesso. Ciascuna entità presente nella federazione SPID è descritta da un file di metadati, che ne riporta il certificato **X509**, gli endpoint e le altre informazioni necessarie alla comunicazione con le altre entità.

Esistono vari *identity provider*, cioè dei gestore di identità digitale. Si occupano di fornire le credenziali[1] di accesso al sistema (identità digitali) e gestire i processi di autenticazione degli utenti. Viene scelto liberamente dall'utente fra quelli disponibili, dunque autorizzati dall'AGID. A questa si affiancano altre due entità, il *Fornitore di servizi* (Service Provider o SP) e il *Gestore di Attributi qualificati* (Attribute Authority o AA) .

In ogni caso, una soluzione basata su SPID prevede la presenza di una TTP.

4.3 La Blockchain come soluzione

Una Blockchain pubblica è una *root of trust* decentralizzata che non viene posseduta da nessuno. Non c'è proprietario, pertanto chiunque può avervi liberamente accesso. La fiducia in un umano, o in una CA, viene sostituita dalla fiducia in nella matematica. Si tratta dunque di un sistema utile per contenere un **registro decentralizzato di chiavi pubbliche**. Una chiave pubblica avrà dunque un proprio indirizzo, chiamato **DID**, decentralized identifier. Dalla PKI, si passa alla **DPKI**, Decentralized PKI.

[1] le credenziali SPID possono avere 3 livelli di sicurezza differenti.

4.3.1 DID e self-sovereign identity

Il **DID** [24], *decentralized identifier* è uno standard W3C per associare ad un'entità generica un ID sotto il controllo del proprietario, **senza l'ausilio di una TTP**. Ogni Blockchain decide poi, eventualmente, in che maniera utilizzarlo. Ogni chiave pubblica ha, grazie alla Blockchain, un proprio indirizzo, appunto un DID, che non richiede alcuna Registration Authority. Nella Blockchain inoltre viene memorizzato un **DID document**, che contiene la chiave pubblica del DID stesso. Il proprietario dell'identità controlla il DID document controllandone la chiave privata. Ogni Blockchain definisce le modalità di scrittura, cioè di *registrazione*, e di risoluzione, dunque di *lettura* di un DID. Risulta necessaria l'applicazione del metodo della *firma digitale*.

L'utilizzo della strategia del *decentralized identifier* dà vita al modello della **self-sovereign identity**. Si tratta di un principio in cui al singolo utente, **proprietario di un wallet apposito**, sia permesso di non delegare la custodia e il controllo delle informazioni personali a terze parti, potendo decidere di esporre solamente i certificati che più ritiene utili. Dunque tale modello garantisce **privacy**, essendo ognuno libero di creare DID a proprio piacimento, ed è fondamentalmente **economico**.

4.3.2 Esempi di integrazione con la Blockchain

SOVRIN

La Blockchain di **Sovrin** [25] è progettata esclusivamente rispetto all'utilizzo di identità digitali. E' pensata per avere la stessa scalabilità dei DNS, basandosi sul *Sovrin Trust Framework*. Dettagliatamente si pone l'obiettivo di garantire le caratteristiche fondamentali di **SSI**, self-sovereign identity, ossia *governance, scalability, accessibility, e privacy*. A differenza dei DNS, è presente un algoritmo di consenso. I nodi della rete si organizzano in *validatori*, che lavorano rispetto a operazioni di scrittura e sono in numero più ristretto, e *nodi observer*, che lavorano rispetto a operazioni di lettura. In tale contesto, si applica l'algoritmo di *State proof*, molto semplice, eseguibile anche su semplici smartphone. E' possibile utilizzare un DID per ogni applicazione. Si pensi che ad Amazon venga dato un codice valido solamente per lui, relativo ad un certo utente e al suo metodo di pagamento. In caso di furto o smarrimento, tale codice non sarà utilizzabile da terzi. Infatti, è importante sottolinera che la blockchain di Sovrin non mantiene dati personali, dunque la privacy viene garantita.

Una possibile problematica è rappresentata dal fatto che i dati criptati su una blockchain pubblica prima o poi verranno violati, a causa della continua evoluzione dei sistemi computazionali, e dunque si è optato per strutturare il sistema con un network P2P parallelo al tipico *ledger*. Tale rete P2P è costituita dagli agenti privati che gestiscono i vari DID. Si utilizzano principi di ZKP e ogni owner è libero di decidere *quali* informazioni condividere.

Infine, si noti che *verificare* una certa identità presenta un certo rischio. Dunque, sulla base di esso, avvengono degli scambi di TOKEN, che vengono *guadagnati* dai *verifier*.

uPort

Il sistema di **uPort** [26] si colloca nell'ambito della Blockchain di Ethereum. Un'identità uPort non è altro che un indirizzo Ethereum. Dunque viene consentito alle applicazioni e ai relativi utenti di scambiare informazioni privatamente, in maniera garantita dal backend Ethereum.

E' dunque più chiaro dire che un'identità uPort [27] è una rappresentazione digitale completa di una persona, un'organizzazione o un dispositivo in grado di interagire con smart contract.

Possiede una serie di caratteristiche che la rendono un'applicazione dalle interessanti potenzialità. In primo luogo, garantisce il principio di *self-sovereign identity*. Permette l'esecuzione di operazioni sia *on-chain* che *off-chain*. Inoltre rende più robusto il sistema, in quanto sono presenti meccanismi di **recovery**. Tali meccanismi sono normalmente complicati, se non impossibili, per sistemi di identità basati su meccanismi asimmetrici, dunque con coppie di chiavi pubblica/privata. Ciò è permesso dal fatto che alcuni smart contract siano in grado di controllarne altri. Il contratto *controller* viene memorizzato in sicurezza nel proprio smartphone. Dunque, un'identità uPort non è altro che un semplice smart contract controllato in tal modo. E' infatti fondamentale l'utilizzo dell'app uPort.

Il **registro uPort** è uno smart contract gestito *on-chain*, condiviso da tutte le identità uPort, che provvede all'infrastruttura richiesta per la condivisione di dati off-chain e per la verifica delle identità. Consente alle entità di fare *affermazioni* sulla propria identità.

Capitolo 5

DTT - Domain Trusted Transaction

5.1 Introduzione

In seguito ad una serie di ricerche, emerge che non vi sono Identity Provider per la certificazione di identità digitali, **riconosciuti su larga scala e globalmente approvati**, che offrano servizi di **integrazione con la blockchain pubblica**.

Per certificare l'associazione tra una transazione e un'identità risulta necessario disporre di un Identity Provider, che sia riconosciuto come *trusted* da tutti i nodi della blockchain nei quali linnformazione è scritta. Nelle Blockchain **private** è possibile utilizzare uno dei tanti Identity Provider già esistenti, purchè vi sia il consenso unanime dei nodi rispetto alla sua autorevolezza.

Per quanto riguarda le Blockchain *consortili*, cioè composte da una serie di enti, è necessario trovare un Identity Provider che sia *super partes*, che abbia pertanto un livello di autorevolezza superiore a ciascun consorziato. Dunque, più sarà marcata la differenza fra i vari enti, più sarà complicato stabilire un Identity Provider sufficientemente autorevole.

5.1.1 La soluzione di ANSA

In realtà una soluzione di integrazione fra *certificazione* e Blockchain è già in fase di sperimentazione per quanto riguarda l'agenzia ANSA [28]. La celebre azienda di informazione multimediale, infatti, con la collaborazione con AY, ha sviluppato un sistema do tracciatura di notizie basato sulla tecnologia Blockchain. Si tratta di una assoluta innovazione per quanto riguarda il mondo dell'editoria. Attraverso il bollino **ANSAcheck** [29] sarà possibile dunque verificare l'origine delle notizie provenienti dai notiziari ANSA, avendo accesso alla storia **notarizzata** di ciascuna notizia, garantendo al lettore la tracciabilità del dato e la trasparenza dell'informazione. Sarà dunque permesso al lettore di verificare l'*autenticità* delle fonti.

Si tratta dunque di un esempio lampante di come la Blockchain applicata all'industria dell'editoria permetta di produrre e distribuire contenuti digitali in maniera affidabile e sicura, aumentando ulteriormente il rapporto di fiducia con il lettore.

La soluzione di ANSA si basa su tecnologia **EY OpsChain Traceability**, caratterizzata da transazioni pubbliche registrate sulla blockchain di Ethereum. Il lettore, visualizzando la notizia sul proprio browser, potrà verificare la presenza o meno del bollino digitale, il quale verrà assegnato dopo i parametri della news pubblicata sul sito ANSA. Solo se tali parametri saranno identici a quelli registrati su blockchain in fase di creazione della news, la notizia riceverà il bollino **ANSAcheck**.

5.1.2 La certificazione delle informazioni

In ogni caso, nell'ambito del progetto PININ, risulta necessaria un'estensione del concetto di certificazione, che non sia limitato al concetto di notizia, ma che sia in grado di certificare un'*informazione* nel senso più generale del termine. In particolare, rispetto alla tecnologia della Blockchain, si intende un'applicazione che possa certificare una transazione, senza, eventualmente, far caso al contenuto, in quanto associata ad un wallet considerato fidato. Nasce dunque l'idea centrale del corrente elaborato, che ha portato allo sviluppo dell'applicativo che si è deciso di denominare **DTT, Domain Trusted Transaction**.

5.2 DTT: valutazioni generali

Per prima cosa occorre dunque ricercare e definire un Identity Provider che sia universalmente riconosciuto e considerato fidato da tutte le entità prendenti parte al sistema di certificazione in questione. In tale contesto, una delle autorità più importanti e *autorevoli*, in assoluto, del web è senza dubbio quella del **DNS**, il Domain Name System.

5.2.1 Il DNS

Il **DNS** [30] è il sistema che permette la traduzione di nomi in indirizzi IP, associando dunque tali nomi ai nodi della rete. Attraverso l'operazione chiamata *risoluzione* è possibile utilizzare i nomi stessi al posto degli indirizzi IP. Il servizio è realizzato mediante un database distribuito, costituito da vari server DNS organizzati secondo una struttura gerarchica ad *albero rovesciato* e si divide in vari domini. Ad ogni dominio è associato un *nameserver*. I nomi di dominio terminano con un "." e la stringa finale rappresenta il *root nameserver*. In particolare, i *root nameserver* sono 13.

La Internet Assigned Numbers Authority (IANA) [31] è responsabile del censimento dei domini di primo livello validi su Internet e degli enti proprietari, sia pubblici che privati. In tale contesto si collocano poi delle aziende dette **registrar**, le quali forniscono servizi di diverso tipo legati alla rete, come l'hosting. Tipicamente vengono dette Service Provider e sulla base di un contratto stipulato con uno o più enti proprietari di domini di primo livello, consentono la registrazione a persone fisiche o giuridiche di un domino **di secondo livello**. Infine, i domini di terzo livello sono gestiti dalla registrar stessa.

Per registrare un dominio è dunque obbligatorio stabilire un **proprietario**, che fornirà alla registrar tutte le informazioni necessarie sulla propria **identità**. Più il proprietario di un host sarà autorevole, più lo saranno i suoi contenuti.

5.3 Descrizione della soluzione

La **Domain Trusted Transaction** è dunque una transazione Blockchain corredata dell'informazione del dominio certificatore. Tale informazione certifica che al momento della scrittura della transazione, il proprietario del dominio indicato, o un suo delegato, fosse proprietario del wallet originatore. L'utilizzo di una soluzione di questo tipo permette di sfruttare tutti quelli che sono i vantaggi della Blockchain, in particolare la possibilità di stabilire, con certezza crittografica, l'integrità dei dati contenuti all'interno della Blockchain.

Sulla Blockchain, se da un lato sussiste la certezza matematica che una transazione sia stata generata dal possessore di un certo wallet, non è possibile risalire all'identità del proprietario del wallet. In generale, l'anonimato degli utenti è una delle principali caratteristiche della Blockchain, ma possono presentarsi casistiche in cui sia l'utente stesso ad aver necessità di vedere *certificata* una determinata informazione, contesto in cui si applicano gli scenari proposti dalla Domain Trusted Transaction.

Lo scenario di applicazione vede le due fasi di certificazione e verifica della certificazione in maniera, ovviamente, ben distinta.

Certificazione

Un generico wallet che sia intenzionato a vedere una certa transazione certificata da un *certo* dominio dovrà rivolgersi ad un apposito contratto sulla **Blockchain di Ethereum** consegnandogli i parametri necessari, ossia *l'hash della transazione da verificare, il dominio di certificazione e*, implicitamente, *il proprio indirizzo*, in modo che questi possa salvarli sulla Blockchain stessa. Tramite un sistema di eventi, inoltre, un server associato al dominio interessato intercetta e memorizza le informazioni del caso, ossia l'hash della transazione e il relativo wallet.

Verifica della certificazione

Un generico utente che invece desideri *verificare* l'autenticità di una certa transazione, si rivolge allo stesso contratto in Blockchain, da cui ottiene *il dominio certificatore e il wallet associato*. Basterà dunque collegarsi al dominio certificatore, verificando la che la transazione sia stata effettivamente certificata dal dominio specificato.

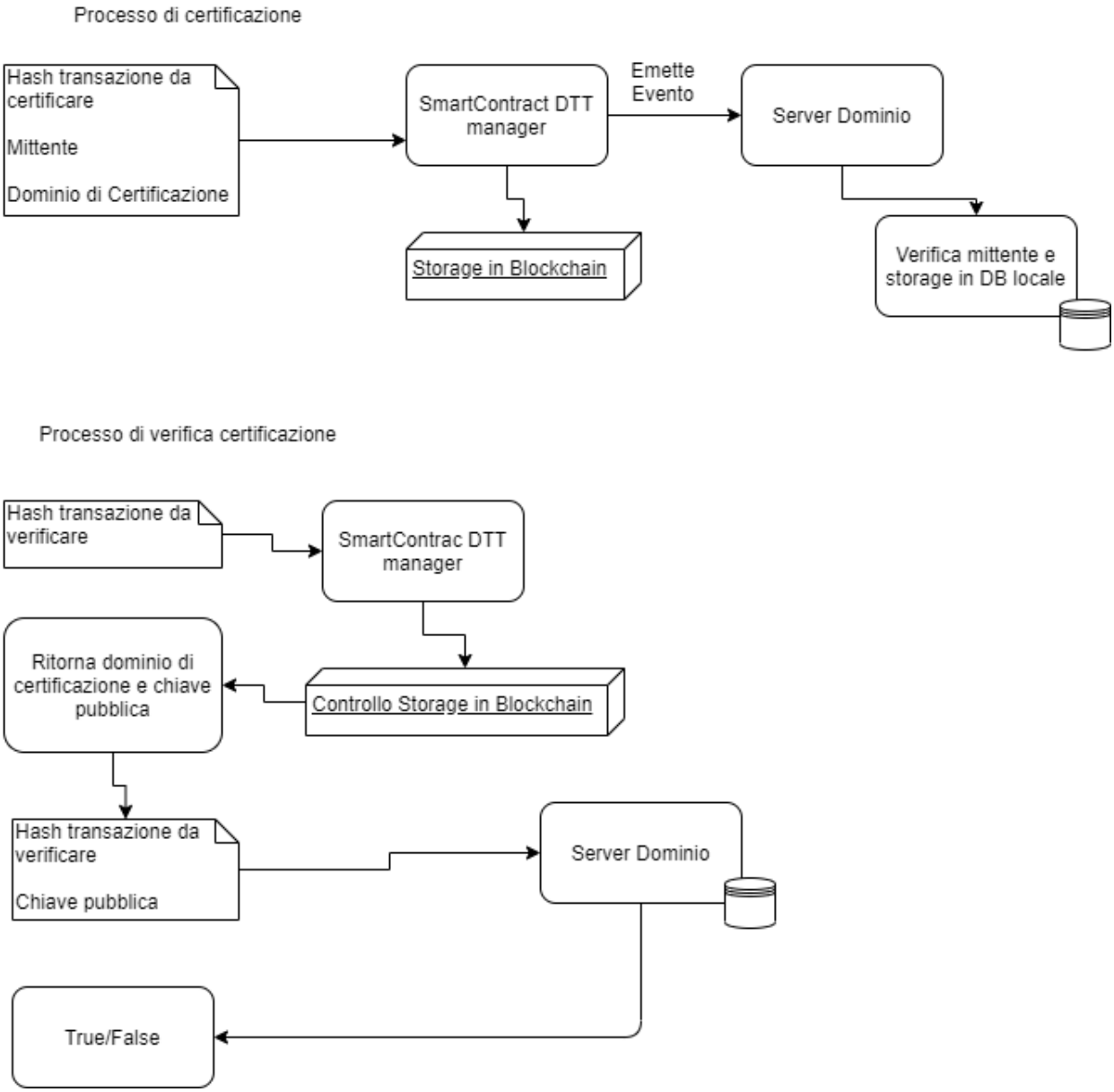

Figura 5.1: Flusso DTT.

5.4 Organizzazione dell'applicativo

5.4.1 Preparazione e strumentazione

Lo sviluppo dell'applicativo del DTT ha previsto l'utilizzo di strumentazioni di diverso tipo, la cui scelta è stata fatta in seguito ad una serie di valutazioni preliminari.

In una prima fase vi sono stati esperimenti riguardo il linguaggio Solidity e deploy di semplici Smart Contract e DApp (decentralized applications), con l'ausilio di semplici editor di testo, dei tool *Ganache* [36], che permette di creare un rete Blockchain locale, e *Truffle* [35], che permette sviluppo e deploy di Smart Contract

e DApp. E' stato anche utilizzato *Metamask* [34], plugin di Firefox utile per connettersi alla Blockchain e interfacciarsi con uno Smart Contract. In questa fase, sono state estremamente utili le guide fornite dal portale web *DApp university* [39].

Una volta acquisita un'adeguata padronanza del linguaggio di programmazione suddetto, si è andato avanti con la creazione e la gestione di una blockchain Ethereum privata in locale, mediante l'ausilio del tool Geth, utile per eseguire transazioni, generare account ed eseguire mining.

Infine, è stato utilizzato l'IDE *Remix* [37] per sviluppare e gestire al meglio lo Smart Contract DTTmanager. Per quando riguarda il *developing* dello stesso, si è fatto uso della rete di test di Rinkeby. In tal contesto sono stati utilizzati i servizi offerti da Infura. Infura [32] è un'infrastruttura che fornisce servizi API per connettersi a una rete Ethereum senza dover istanziare un nodo Ethereum e dunque senza dover scaricare l'intera blockchain. È un servizio molto comodo e veloce che ha come svantaggio la perdita di decentralizzazione, in quanto tutte le operazioni sono gestite da un servizio esterno.

L'interazione con lo Smart Contract anche per entità esterne alla Blockchain stessa è permessa dalla liberia **Web3** [33]. Lato server infatti, è stato implementato uno script basato sul framework *node.js* [38], il quale si integra perfettamente con la suddetta libreria.

In fase di verifica della certificazione, si è sviluppata una semplice pagina web, utilizzando i linguaggi HTML, PHP e Javascript, la quale interagisce con il Database in questione, costituendo l'effettivo *front-end* dell'applicativo sviluppato.

5.4.2 Sviluppo

Fondamentalmente, i processi in questione avvengono in maniera distinta ed indipendente tanto su Blockchain quanto dal punto di vista del server di dominio. Si è scelta una soluzione di questo tipo per garantire la massima **decentralizzazione** possibile al sistema sviluppato.

5.4.3 Processo di richiesta di certificazione

- Smart Contract centrale **DTTmanager**. Si tratta del contratto fulcro dell'applicativo pensato. Tiene memoria di tutte le transazioni finora certificate, mettendole in relazione con il relativo dominio di certificazione. I contratti che vogliono certificare una certa transazione invocano dunque tale contratto, passandovi i parametri del caso (HASH della transazione da certificare, dominio di certificazione). Una volta passati i parametri, si richiede la certificazione effettiva. Il DTTmanager emette dunque un **evento**, che verrà intercettato poi dal server del dominio al quale si richiede la certificazione.

- Lato *server* vi è uno script che è in continuo ascolto degli eventi emessi dal contratto DTTmanager. Al suo avvio, tale script scarica tutti gli eventi passati, filtrandoli e verificando di non aver perso eventi. Il server sarà connesso ad un DB locale, in cui collocare tutte le transazioni certificate. In caso di

approvazione della certificazione, dunque dopo aver intercettato un *evento* a lui riferito firmato da un wallet autorizzato, il server memorizza la transazione in questione. In particolare, il server tiene nel proprio DB un elenco delle transazioni attualmente certificate e dei wallet attualmente autorizzati. Lo script in questione si pone in ascolto di tutti gli eventi emessi dal contratto DTTmanager, prendendo in considerazione soltanto quelli riferiti al proprio dominio.

5.4.4 Processo di verifica della certificazione

- Data una transazione che si intende verificare, la si passa al contratto DTTmanager, il quale, se la stessa è presente nel proprio storage, ritornerà il dominio che la ha certificata e il wallet associato.

- Collegandosi al dominio ricevuto, si andrà a verificare che la transazione sia stata effettivamente certificata. Occorrerà dunque inserire sia l'hash della transazione che la chiave pubblica del wallet associato, in modo che il server possa eseguire una firma *real-time*, dimostrando non solo di aver certificato la transazione, ma anche di possedere la chiave privata del wallet in questione.

5.4.5 Organizzazione del DB

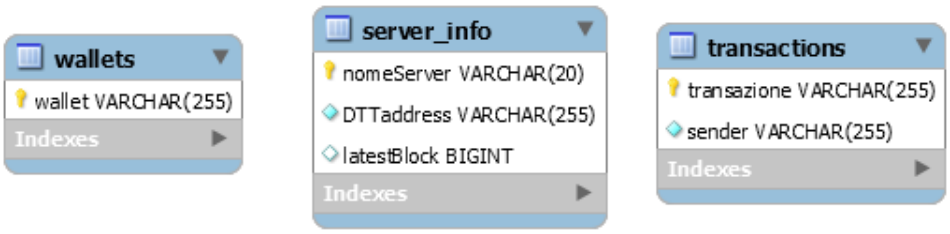

Figura 5.2: Organizzazione del DB.

Per tenere traccia degli indirizzi autorizzati, delle transazioni certificate e di alcune informazioni di servizio, il server gestisce un semplice database MYSQL, organizzato secondo le seguenti tabelle[1]:

- server_info(nomeServer, DTTaddress, latestBlock). Tabella in cui si tiene memoria del nome del dominio in questione, dell'indirizzo Ethereum cui il contratto DTTmanager è stato *deployato* e del numero dell'ultimo blocco della Blockchain analizzato dal server. Si noti che al suo avvio, il server verifica tutti gli eventi passati, in modo da evitare di perderne. Dunque il parametro `latestBlock` serve per evitare di processare più volte gli stessi blocchi.

[1] le chiavi primarie sono sottolineate

- wallets(<u>wallet</u>). Tabella in cui si tiene memoria delle chiavi pubbliche dei wallet correntemente autorizzati.
- transaction(<u>transazione</u>, wallet). Tabella in cui si tiene memoria di ciascuna transazione certificata insieme al relativo wallet.

5.5 Implementazione

5.5.1 DTTmanager

Il contratto DTTmanager costituisce a tutti gli effetti il cuore del Domain Trusted Transaction, occupandosi di memorizzare su Blockchain tutte le informazioni necessarie al fine di certificare una transazione ed eventualmente verificarne la certificazione. Inoltre, mediante l'emissione di eventi, comunica con i vari domini, i quali procederanno poi in maniera indipendente nel gestire le certificazione, in un secondo momento. Le strutture fondamentali sono due.

In primo luogo va sottolineata la definizione dell'evento `Certification`, che contiene i *parametri di certificazione*, ossia il domino di certificazione, l'indirizzo del mittente, un ID univoco[2] e ovviamente l'hash della transazione da verificare. Tale evento è **fondamentale ed imprescindibile** ai fini del funzionamento generale del programma. Infatti solo leggendo il campo `certification_domain` un server di dominio sarà in grado di capire quali sono gli eventi effettivamente a lui destinati.

```
180     event Certification(
181         string certification_domain,  //Dominio di certificazione
182         address sender,   //Soggetto che richiede la certificazione
183         uint id,     //(Random) ID della certificazione
184         string transaction_certified  //Transazione certificata
185     );
```

Listing 5.1 certificationEvent - da DTTmanager.sol

In secondo luogo va fatta menzione della struttura che permette lo *storage* delle informazioni, `certificationList`. Si tratta di un *mapping*, che associa a ciascun *hash di transazione* una struttura dati `Certificate`, contenente il nome del dominio di certificazione e il wallet associato alla transazione.

La funzione `addCertification` si occupa invece di *emettere* l'evento `Certification`, salvando poi i dati relativi nel proprio storage. Si noti che pur provando ad emettere un evento utilizzando dati falsi, ad esempio designando un dominio presso il quale non si è autorizzati, questi sarà ignorato dal server.

```
35      function addCertification(string memory _transaction, string memory
            _domain) public returns(bool){
36          if(!certificationList[_transaction].present){
```

[2] nel corso dello sviluppo l'utilizzo di questo parametro è stato omesso, ma si è preferito non modificare la struttura dati in vista di eventuali sviluppi futuri del progetto.

```
37          certificationList[_transaction] =Certificate(true, _domain,
                msg.sender);
38          totalCertifications++;
39          emit Certification(_domain, msg.sender, totalCertifications,
                _transaction);
40          return true;
41          }
42          else
43          return false;
44      }
```

Listing 5.2 certificationEvent - da DTTmanager.sol

5.5.2 Server

Lato server, gli eventi emessi dal DTTmanager vengono dunque intercettati e gestiti da uno script sviluppato mediante il framework *node.js*.

Tale script va in prima battuta configurato. Affinchè funzioni correttamente, occorre impostare:

- Il nome del dominio, necessario anche per la connessione al DB.
- I parametri di connessione al DB locale.
- I parametri di connessione allo Smart Contract DTTmanager, specificandone l'ABI e l'indirizzo Ethereum.

Il flusso principale dello script viene specificato nella funzione `main`. Si noti che è stato fatto un largo utilizzo degli oggetti **Promise**. Gli oggetti `Promise` sono usati per computazioni in differita e asincrone. Una Promise rappresenta un'operazione che non è ancora completata, ma lo sarà in futuro. Una soluzione di questo tipo era necessaria per potere gestire in maniera sequenziale l'accesso al DB, per evitare errori. Ad esempio il server potrebbe ricevere due eventi distinti con lo stesso Hash di transazione. In tal caso solo il primo arrivato andrebbe memorizzato in locale, mentre il secondo andrebbe ignorato. Per evitare problemi di sincronizzazione di ogni tipo, il DB viene dunque gestito, a livello di codice, mediante il paradigma delle Promise.

La funzione `checkPastEvents` si occupa di verificare eventuali perdire di eventi passati, mentre la funzione `listen` si occupa di mettersi in ascolto di eventi in tempo reale.

```
129 function checkPastEvent(event){
130   var promise =new Promise(function(resolve, reject){
131     setMax(event.blockNumber);                //Blocco in cui si trova
            l'evento, si controlla che sia o meno il massimo
132     if( event.returnValues[0] ===currentDomain AND
            event.returnValues[3].length >0){   //Rilevato dominio corretto -
            Verifico che sia transazione non nulla
133       var currentTransaction =event.returnValues[3];          //Estrazione
              della transazione da certificare
```

```
134       var sender =event.returnValues[1];              //Estrazione del wallet
              certificatore
135       checkOldTransaction(currentTransaction).then(function (l) {
              //Controllo che la transazione non sia presente nel DB
136         if( l ===currentTransaction ){
137           checkAuthWallet(sender).then(function (s) {         //Verifica che
                  il wallett sia autorizzato
138             if(s ===1) {
139               console.log('-----------------------------------------------------');
140                                               //Il server ha perso l'evento,
                                                      aggiorna il DB
141               console.log(
142               'Recupero transazione ' +currentTransaction +'...\n' +
143               'Evento di certificazione rilevato per il dominio: ' +
                      currentDomain +'\n' +
144               'Mittente: ' +sender
145               );
146               addTransaction(currentTransaction, sender).then(function (_add) {
                       //Esecuzione query di aggiornamento DB
147                 //console.log(\_add);

149                 if(_add ===currentTransaction){
150                   resolve("Transaction_exec");        //Query eseguita
                          correttamente
151                 }
152                 else
153                   resolve("Transaction_reject");         //La transazione era
                          gia' presente e non e' stata inserita

155                 }).catch(function (err) {
156                 console.log("Error " +err);
157                 reject(err);
158                 });
159             }
160             else
161               resolve("Unauthorized_wallet!");

163           }).catch(function (err) {
164             console.log("Error Check Wallet " +err);
165             reject(err);
166           });

168         }
169         else
170           resolve("Already_present_TX");
171       }).catch(function (err) {
172         console.log("Error Check old_trans" +err);
173         reject(err);
174       });
175     }else{
176       //Dominio differente
177       resolve("Diff_domain");
178     }


181   });
182   return promise;
```

```
183 }
```

Listing 5.3 checkPastEvent - da Server.js

```
270 function listen(){
271   console.log('-----------------------------------------------------');
272   console.log('Server in ascolto...');
273     //Il server si mette in ascolto degli eventi
274     contract.events.Certification()
275     .on('data', (event) =>{
276       //console.log(event);
277       var certification_domain =event.returnValues[0];      //Estrazione
              dominio di certificazione
278       if(currentDomain ===certification_domain ){       //Verifica del dominio
279         var currentTransaction =event.returnValues[3];      //Estrazione
              della transazione da certificare
280         var sender =event.returnValues[1];         //Estrazione del wallet
              certificatore
281           checkAuthWallet(sender).then(function (s) {      //Verifico che il
                wallet sia autorizzato
282             if(s ===1) {
283               console.log('-----------------------------------------------------');
284               console.log('Evento di certificazione rilevato per il dominio: '
                    +certification_domain +'\n' +
285               'Mittente: ' +sender +"\nTransazione: " +currentTransaction ) ;

287               console.log('Server in ascolto...');
288               addTransaction(currentTransaction, sender).then(function (_add) {
                      //Esecuzione query di aggiornamento DB
289               // console.log(_add);
290                 }).catch(function (err) {
291                 console.log("INSERT error", err);
292                 });
293               //addTransaction(events[i].returnValues[3],
                    events[i].returnValues[1]);
294             }
295             else{
296               //console.log("L'indirizzo rilevato non e' autorizzato sul
                    dominio " + currentDomain + '\n\n');
297             }
298             }).catch(function (err) {
299             console.log("Error", err);

301             });
302           //var s = checkAuthWallet(sender);
303       }
304       else {
305         //console.log('Dominio errato \n\n');
306       }
307     })
308     .on('error', console.error);
309 }
```

Listing 5.4 listen - da Server.js

5.5.3 Verificatore DTT

La fase di *verifica* della certificazione può essere vista come l'effettivo *front-end* dell'applicativo DTT. Nel dettaglio, è stata implementata una semplice pagina web, che si occupa di:

- Acquisire le informazioni necessarie alla verifica di una certa transazione.
- Generare un firma digitale.
- Verificare la firma.

Inoltre sono presenti una serie di messaggi maggiormente esplicativi, espressi mediante appositi tooltip, volti a definire più nel dettaglio il significato di ciascun elemento della pagina web.

Verifica transazione

In particolare, è presente un form in cui l'utente dovrà introdurre obbligatoriamente l'hash della transazione in questione e la chiave pubblica del wallet associato. In tal caso, il sistema verifica che la transazione sia effettivamente presente nel DB e che sia associata al wallet inserito, ritornando eventualmente un messaggio di errore.

```
1  function verifyTransaction()
2  {
3    $conn =dbConnect();
4    $transrec =mysqli_real_escape_string($conn, $_POST['transazione']);
5    $transazione= mysqli_query($conn, "SELECT transazione FROM
         transactions WHERE transazione = '".$transrec."'");
6    $trans =mysqli_fetch_array($transazione);
7    $tr =$trans[0];

9    if (!$transazione)
10   {
11     echo /*Codice HTML error*/;
12   }
13   else
14   {
15     //Se la transazione e' presente ne si estrare il wallet di
           certificazione
16     $kPub =checkTransactionDomain($_POST['transazione']);
17     $array =array($kPub);
18     $sender_kPub =implode("" ,$array);
19     $signature ="";
20     $today ="";
21     if($sender_kPub ===$_POST['kPub']){
22       //La chiave pubblica inserita corrisponde a quella associata alla
             transazione nel DB
23       $current_timestamp =time();
24       $today =date("D M j G:i:s T Y",$current_timestamp). "</br>";
```

```
25        //Viene eseguito lo script generateSignature.js che permette di
              calcolare la firma digitale del dato wallet, avendo
26        //accesso alla sua chiave privata. La firma viene calcolata su un
              messaggio composto dalla transazione stessa
27        //seguito dal timestamp attuale.
28        $exec ='node generateSignature.js ' . $tr . " " .
              $current_timestamp . " ". $sender_kPub ;
29        $escaped_command =escapeshellcmd($exec);
30        $signature =exec($escaped_command);
31        $parameters =
              "'".$signature."','".$sender_kPub."','".$tr."','".$current_timestamp."'";
32        //Viene eseguita lato cliente una verifica della firma
33        echo '<script> verifySignature(' .$parameters. '); </script>';
34      }
35      else
36        echo"<script>errorSender();</script>";

38    }
39    mysqli_free_result($transazione);
40    //Stampa dei parametri in caso di esito positivo
41    if($kPub !=null){
42    echo /*Codice HTML presentazione dati*/;

44    }

46    else  //La transazione inserita dall'utente non e' stata trovata nel
          DB
47    {
48      echo /*Codice HTML error*/
49      ;
50    }

52  }

54  if(isset($_POST['transazione']))
55  {
56     verifyTransaction();
57  }

59  ?>
```

Listing 5.5 VerifyTransaction.php

Generazione firma

La firma digitale viene generata mediante la funzione `sign` offerta dalla libreria Web3 su un messaggio composto dalla concatenazione dell'hash della transazione e un TimeStamp. Il server, in locale, ha infatti accesso alle **chiavi private** dei vari wallet, senza le quali sarebbe ovviamente impossibile procedere con il sistema della

firma digitale. Nel dettaglio, si riporta lo script node che esegue la firma. Si noti che, chiaramente, la firma viene eseguita lato server.

```
2  //Script in grado di calcolare firma digitale su un dato messaggio.
3  //Ha accesso alla chiavi private dei wallet autorizzati

5  var Web3 =require('web3');
6  const web3 =new Web3();
7  //Transaction hash + timestamp
8  let data =process.argv[2] +" " +process.argv[3];

10 var kPub =process.argv[4];

12 var fs =require("fs");
13 //Il nome del file e' la chiave pubblica, il contenuto e' la chiave
       private
14 var privateKey =
       fs.readFileSync("./keystore/"+kPub+".txt").toString('utf-8');
15 privateKey =privateKey.trim();

17 var res =web3.eth.accounts.sign(data, privateKey);

19 console.log(res['signature']); //Ritorno valore a script chiamante

21 process.exit();
```

Listing 5.6 generateSignature.js

Verifica firma

In ultima battuta, viene eseguita una funzione JavaScript, lato client stavolta, che verifica la validità della firma appena generata, in modo da dimostrare all'utente che sta visitando la pagina di essere in possesso della chiave privata del wallet in questione. In particolare, viene eseguita la funzione `verifySignature`, che utilizza la funzione `recover` messa a disposizione dalla libreria Web3.

```
3  /**
4  * @desc Funzione eseguita lato Client. Si occupa di verificare una
       firma digitale, mediante l'utilizzo della funzione recover(), messa
       a disposione dalla libreria Web3
5  * @param string signature firma digitale da verificare
6  * @param string kPub chiave pubblica
7  * @param string txHash della transazione
8  * @param string ts TimeStamp di generazione della firma
9  */
10 function verifySignature(signature, kPub, txHash, ts){
```

```
12 const web3 =new Web3();

14 //Ricostruzione del messaggio firmato
15 var message =txHash +" " +ts;
16 //La funzione recover(), messa a disposione dalla libreria Web3,
       permette di ottenere la chiave pubblica di una data firma digitale
17 //avendone ricevuto in input il messaggio firmato e la relativa firma
18 var dec =web3.eth.accounts.recover(message, signature);
19 if(dec ===kPub)
20   {
21     //alert("Verifica firma effettuata correttamente. Chiave pubblica
           risultante: " + dec);
22     document.getElementById('failSign').style.display='none';
23     document.getElementById('okSign').style.display='block';
24   }
25 else
26   errorSender();
27 }
28 function errorSender(){
29   //alert("Non e' stato possibile generare una firma. Per favore,
         inserire valori corretti.");
30     document.getElementById('failSign').style.display='block';
31     document.getElementById('okSign').style.display='none';
32 }
```

Listing 5.7 verifySignature.js

5.5.4 Note

Si noti che, nel corso dell'elaborato, si sono riportati solamente i pezzi di codice considerati maggiormente rilevanti. In ogni caso nella sezione Appendice è presente il codice nella sua interezza.

Rispetto al codice della pagina web di verifica sono state riportate solo le funzioni più significative, omettendo il codice HTML relativo alla grafica della pagina e le annesse funzioni JavaScript. Inoltre va evidenziato che la pagina web è stata implementata prevalentemente utilizzando il linguaggio PHP per l'interazione con il DB, mentre la generazione della firma, che avviene comunque lato server, è scritta su node.

Di seguito si riportano alcuni esempi della pagina web in questione.

Figura 5.3: Pagina iniziale DTT *verifier*.

Figura 5.4: Transazione correttamente verificata e certificata.

Figura 5.5: Transazione assente.

Capitolo 6

Conclusioni

In conclusione, l'applicativo prodotto, pur essendo stato sviluppato sotto una prospettiva prettamente didattico-sperimentale, risulta funzionante e coerente con i requisiti richiesti. Nonostante concepito nell'ambito del sopracitato progetto PININ, si tratta di un applicativo estremamente modulare, applicabile a qualsiasi contesto in cui sia necessaria la certificazione delle informazioni mediante tecnologia Blockchain.

Nell'ambito del progetto PININ, dunque, il DTT potrà essere una soluzione utile a tutte le parti in causa per la garanzia delle varie informazioni. Gli allevatori avranno modo di caricare le proprie informazioni su un sistema sicuro, con la garanzia che queste non vengano manomesse o falsificate, mentre un generico consumatore potrà verificare liberamente le informazioni sulla filiera.

6.1 Analisi sperimentale

Il progetto PININ, basato fondamentalmente su una tecnologia decentralizzata e distribuita, che vede l'anonimato come caratteristica fondamentale, non poteva prescindere dalla presenza di una strategia che permettesse la certificazione delle informazioni in gioco. Difatti, si trattava di un progetto per molti aspetti ancora in fase di progettazione piuttosto che in fase di sviluppo.

Difficile, infatti, pensare ad un sistema che non fosse in grado di verificare l'autenticità dei dati che sarebbero poi stati diffusi fra i consumatori. Poiché il progetto è stato pensato per *girare* su una Blockchain permissionless, era necessaria una strategia innovativa. Anche in seguito ad una serie di ricerche, non è stata trovata nessuna soluzione adeguata già esistente che si potesse applicare alla problematica presentata, risultando dunque necessario elaborare una proposta *ad-hoc*.

Le soluzioni già esistenti sono state scartate per vari motivi. La soluzione di Sovrin, ad esempio, prevedeva l'utlizzo di una Blockchain privata, in contrasto con l'idea di utilizzare la Blockchain pubblica di Ethereum. In secondo luogo, la soluzione di SPID risultava inadeguata in quanto non integrabile adeguatamente con la Blockchain. Infine la soluzione di uPort, per quanto perfettamente integrata con la tecnologia di Ethereum, non possedeva riferimenti a membri esterni alla

Blockchain, in cui veniva dunque racchiuso l'intero processo di certificazione. In grembo a questa necessità nasce l'idea della DTT.

La soluzione è stata discussa durante una serie di riunioni, attraverso le quali è stato possibile definirne le specifiche e i vari casi d'uso. Rispetto all'effettiva integrazione della soluzione nel progetto PININ, i tutor aziendali che più da vicino hanno seguito lo sviluppo hanno approvato il lavoro e lo proporranno nelle prossime riunioni quale soluzione al problema.

6.1.1 Dati sperimentali

Il contesto reale di applicazione è molto simile a quello di test. L'intero sistema è stato sviluppato sulla rete di test di Rinkeby per quanto riguarda il lato Smart Contract e simulando un server in locale per quanto riguarda il *developing* della pagina web. Si tratta di ambienti estremamente simili ai contesti reali, che funzionerebbero esattamente alla stessa maniera tanto sulla Blockchain di Ethereum quanto su un webserver online.

I dati utilizzati in fase di test sono hash di transazioni e coppie di indirizzi *reali*, dunque la soluzione, se considerata sotto un'ottica *non-enterprise*, risulta già applicabile a contesti reali.

6.1.2 Benefici e problematiche

Tabella 6.1: Confronto fra DTT e soluzioni centralizzate, in aggiunta alle riflessioni già espresse.

DTT	**Soluzione centralizzata**
Necessaria sicurezza su ogni dominio	Unico sistema da proteggere
Non ci sono regolamentazioni interne.	Maggiore trasparenza nel rapporto con i singoli utenti
Self-sovereign identity	La gestione dell'informazione non è possibile senza la mediazione dell'ente centrale.
Complessità di setup, configurazione e gestione distribuita fra i domini.	La complessità è interamente a carico dell'ente centrale, dunque maggiormente *user-friendly*

In merito alla certificazione delle informazioni, la soluzione proposta prevede una approccio completamente diverso rispetto alle modalità più *tradizionali*. Basti pensare al concetto di identità. Normalmente esiste un ente considerato *super partes*, come un'autorità statale, che rilascia documenti di identità, garantendone la validità. In un contesto più informatico, le Certification Authority si occupano di rilasciare dei certificati di identità. L'approccio della DTT, invece, elimina del tutto il ruolo di queste entità. La fiducia viene interamente spostata sulla gerarchia

del DNS, tramite la quale è possibile stabilire la proprietà di un certo dominio. Ogni utente che intende certificare una certa informazione sarà identificato dal dominio sotto il quale questi la pubblica, senza dover necessariamente specificare informazioni personali. Dunque, perlomeno in linea teorica, viene garantita una certa privacy al singolo utilizzatore. In caso di violazioni, tuttavia, il dominio stesso sarà in grado di risalire a chi ha effettuato e richiesto la certificazione di una certa informazione. Per certi versi, si tratta di un esempio di *self-sovereign identity*, in cui, se ogni dominio è considerato *possessore* dei wallet ad esso riferito, questi sarà l'unico in grado di risalire agli effettivi utenti che hanno eseguito le varie operazioni. Il principio di anonimato della Blockchain non viene del tutto stravolto e la decentralizzazione non è tuttavia *assoluta*. In primo luogo, ogni utente sarà identificato da un dominio, il quale si farà garante di un sottogruppo. In secondo luogo ciascun utente non risulta del tutto indipendente, ma in un certo senso *legato* ad un suo "ente superiore", ossia il suo dominio di riferimento.

Una possibile problematica può nascere dal fatto che, internamente, un dominio non deve sottostare ad alcuna regola. Pertanto potrebbero verificarsi episodi discriminatori all'interno del dominio, prediligendo alcuni utenti rispetto ad altri nell'approvazione delle certificazioni. Inoltre, server di dominio sono chiaramente più vulnerabili rispetto a server di enti statali o comunque *centrali*. Ciò genererebbe problemi di sicurezza, che richiederebbe a ciascun dominio di dotarsi di un adeguato sistema di protezione.

Si pensi al progetto PININ. In un ambiente che vede la cooperazione fra enti paritari, la violazione anche di uno solo fra i soggetti in questione comporterebbe conseguenze, più o meno gravi, su tutti gli altri.

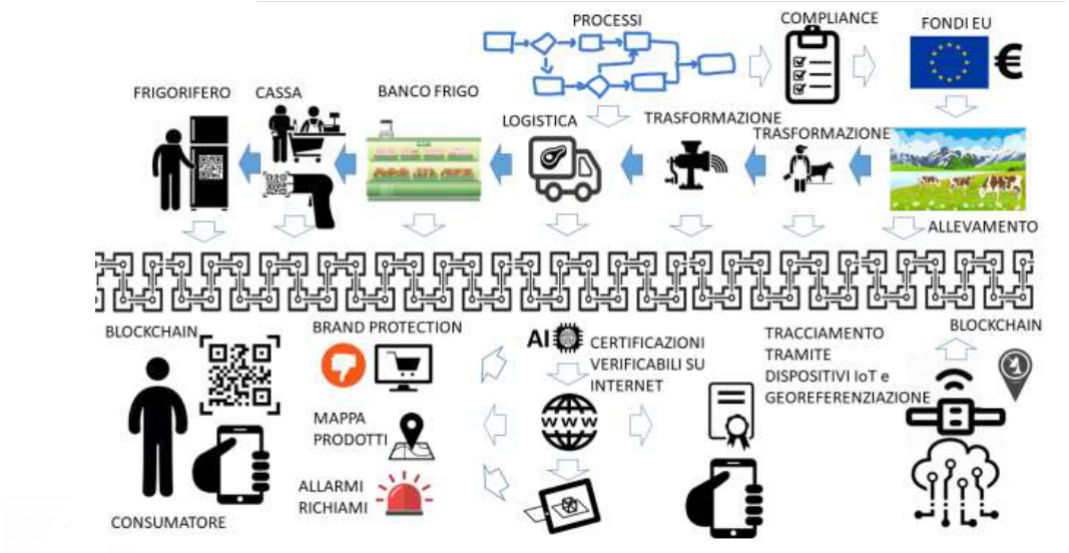

Figura 6.1: Schema concettuale del progetto PININ.

6.2 Evoluzioni future

L'ottica sotto la quale tale programma è stato sviluppato è chiaramente limitata se osservata sotto un punto di vista prettamente aziendale. Si tratta di una versione

sviluppata in sotto un'ottica, appunto, didattico-sperimentale. Una possibile evoluzione del programma verso una versione maggiormente **Enterprise** richiederebbe un miglioramento, per prima cosa, delle caratteristiche di robustezza e sicurezza del sistema sviluppato. Ad esempio, sarebbe opportuno uniformare il linguaggio utilizzato, in quanto nella fase di verifica si fa uso di una combinazione di JavaScript e PHP. Inoltre, le chiavi private sono salvate in locale sul filesystem del server, mentre sarebbe opportuno salvarle mediante un *keystore*.

Sarebbe inoltre opportuno fornire tanto lo Smart Contract **DTTmanager** quanto lo script **Server** di un'interfaccia grafica che ne semplifichi la gestione e il monitoraggio, rendendo l'intero programma maggiormente *user-friendly*.

Il sistema della DTT consente di dimostrare che **al momento della generazione della transazione** certificata il dominio certificatore fosse il proprietario del wallet che l'ha emessa. Una possibile evoluzione può consistere nella verifica **attuale** della proprietà del wallet, non solo certificando che al momento della transazione il wallet fosse di proprietà del dominio dichiarato. Per provare che la firma sia eseguita "al momento", dimostrando quindi la proprietà del wallet attuale, il client dovrebbe effettuare la richiesta passando un token di sfida, come un hash crittografico random generato al momento, o in ogni caso qualcosa di arbitrario scelto dal client, e il server dovrebbe includere questa informazione tra i dati firmati.

Finora si è parlato di certificazioni riguardanti fondamentalmente *transazioni* generiche. Nel caso in cui la certificazione riguardasse persone fisiche si potrebbe modificare il sistema in modo da integrare una possibile funzione di **posta elettronica certificata**.

Capitolo 7

Appendice

7.1 DTTmanger.sol

```
2  pragma solidity >=0.4.21<0.7.0;

4  /** @title DTT manger*/
5  contract DTTmanager{
6      mapping(string =>Certificate) public certificationList;
7      mapping(uint =>string) public certificationDomains;
8      uint public domainsNumber =0;
9      uint public totalCertifications =0;
10     string public absent ='absent' ;

12     struct Certificate{
13         bool present;
14         string domain;
15         address sender;
16     }

18      event Certification(
19         string certification_domain,  //Dominio di certificazione
20         address sender,   //Soggetto che richiede la certificazione
21         uint id,     //(Random) ID della certificazione
22         string transaction_certified  //Transazione certificata
23     );

25     /**
26     * @dev Funzione che aggiunge una transazione all'elenco delle
27     * certificazioni di un relativo dominio.
28     * La transazione viene aggiunta solo se non presente nel mapping,
29     * dunque non e' lecito sovrascriverle
30     * Emette un evento, diretto al server di dominio, che si occupera'
           di
31     * verificare l'autenticita' di una transazione
32     * @param _transaction transazione da certificare
```

```
33      * @param _domain domino di certificazione
34      */
35      function addCertification(string memory _transaction, string memory
            _domain) public returns(bool){
36          if(!certificationList[_transaction].present){
37          certificationList[_transaction] =Certificate(true, _domain,
                msg.sender);
38          totalCertifications++;
39          emit Certification(_domain, msg.sender, totalCertifications,
                _transaction);
40          return true;
41          }
42          else
43          return false;
44      }
45      /**
46      * @dev Funzione che permette di aggiungere dominio alla lista di
            domini
47      * certificatori
48      */
49      function addDomain(string memory _domain) public{
50          certificationDomains[domainsNumber] =_domain;
51          domainsNumber++;
52      }

54      /**
55      * @dev data una transazione, ne si ritorna il dominio di
            certificazione
56      * @param _transaction, transazione di cui si intende ottenere il
            dominio
57      * @return string dominio di certificazione o 'absent' se la
            transazione
58      * non e' presente nel mapping
59      */
60      function getCertDomain(string memory _transaction) public view
            returns(string memory){
61          if(certificationList[_transaction].present ==true)
62              return certificationList[_transaction].domain;
63          else
64              return absent;
65      }
66       /**
67      * @dev data una transazione, ne si ritorna il wallet di
            certificazione
68      * @param _transaction, transazione di cui si intende ottenere il
            wallet
69      * @return string dominio di certificazione o nulla se la transazione
70      * non e' presente nel mapping
71      */
72      function getCertWallet(string memory _transaction) public view
            returns(address){
```

```
73          if(certificationList[_transaction].present ==true)
74              return certificationList[_transaction].sender;

76      }

78  }
```

Listing 7.1 DTTmanager

7.2 Server.js

```
2  /*==============================================================================
3                      DTT Server
4          Programma di ascolto e gestione degli eventi emessi da uno
               SmartContract.
5          1. Configurazione connessione a DB locale e a Smart Contract
6          2. Recupero di eventuali eventi passati non gestiti
7          3. Ascolto di eventi live

9  ==============================================================================*/

11 /*
12   Struttura dell'evento certificatore emesso dallo Smart Contract
         DTTManager
13       event Certification(
14        string certification_domain, //Dominio di certificazione
15        address sender, //Soggetto che richiede la certificazione
16        uint id, //ID incrementale della certificazione
17        string transaction_certified //Transazione certificata
18      );

20 */

22 /*Configurazione parametri necessari per l'utilizzo della libreria Web3
       */
23 var mysql =require('mysql');
24 const Web3 =require('web3');
25 const web3 =new
       Web3('wss://rinkeby.infura.io/ws/v3/cd8f7e5a6b24422fbb3920713c21be27');

27 /*Configurazione dello Smart Contract DTTmanager, il quale si occupera'
       di emettere gli eventi di certificazione e di
28 mantenere in Blockchain lo storico delle transazioni certificate*/
29 const abi =/*ABI Smart Contract DTTmanager*/;
30 const address =/*Indirizzo Smart Contract DTTmanager*/;
31 const contract =new web3.eth.Contract(abi, address);
32 /*==============================================================================*/
```

```
34 var lastAnalyzedBlock;     //Ultimo blocco analizzato
35 var max_block_tmp;

37 var dbname ="Polito" ;    //Nome del Database locale
38 var currentDomain =dbname +".it";  //Nome del Dominio corrente
39 var con;

41 /*Programma principale*/
42 main();

44 /**
45 * @desc Controllo se una transazione sia presente o meno del DB
46 * @param string _transaction, transazione di cui si intende verificare
     la presenza
47 * @return string|bool, _transaction se la transazione non e' presente,
     false se la transazione e' presente
48 */
49 async function checkOldTransaction(_transaction){
50   var promise =new Promise(function (resolve,reject) {
51     con.query("SELECT transazione FROM transactions WHERE transazione =
         '" +_transaction +"'", function (err, result, fields) {
52       //console.log( \_transaction, result.length);
53       if (err)
54         reject(err);
55       else if (result.length ===0)
56         resolve(_transaction) ;
57       else
58         resolve(false);
59       });
60   });
61   await promise;
62   return promise;
63 }

65 /**
66 * @desc Controllo se un wallet e' autorizzato o meno dal dominio
     corrente
67 * @param string _sender, indirizzo di cui si intende verificare
     l'autorizzazione
68 * @returns int, risultato query, 1 se wallet presente, 0 se wallet
     assente
69 */
70 function checkAuthWallet(_sender){
71   var promise =new Promise(function (resolve,reject) {
72     var currentDomain =dbname +".it";
73     con.query("SELECT wallet FROM wallets WHERE wallet = '" +_sender +
         "'", function (err, result, fields) {
74       if (err)
75         reject(err);
76       else if (result)
```

```
77          resolve(result.length) ;
78        });
79    });
80    return promise;
81  }

83  /**
84  * @desc Aggiunta di una transazione al DB. Prima di eseguire la query,
        si esegue un ulteriore controllo rispetto alla presenza o meno
        della transazione
85  * @param string _transaction, transazione da aggiungere e
86  * @param string _sender relativo wallet di certificazione
87  * @return string|bool, _transaction se la query e' andata a buon fine,
        false se e' fallita
88  */
89  async function addTransaction(_transaction, _sender){
90    var promise =new Promise(function (resolve,reject) {
91      checkOldTransaction(_transaction).then(function(rec){
92        if(rec ===_transaction)
93        {
94          var sql ="INSERT INTO transactions (transazione, sender) VALUES
              ('" +_transaction +"','" +_sender +"')";
95          con.query(sql, function (err, result) {
96            if (err)
97              reject(err);
98            else if (result)
99              resolve(_transaction);
100           else
101             resolve(false);

103         });
104         resolve(_transaction);
105       }
106         else
107           resolve(false);
108     }).catch(function(err){
109       console.log("Error" +err);
110       reject(err);
111     });

113   });
114   await promise;
115   return promise;
116 }
117 /**
118 * @desc Funzione per calcolare l'ultimo blocco analizzato
119 * @param int curr valore blocco corrente
120 */
121 function setMax(curr){
122   if (curr >max_block_tmp)
123     max_block_tmp =curr;
```

```
124 }
125 /**
126 * @desc Funzione che si occupa di verificare se un dato evento e' stato
      perso
127 * @param event, evento che si intende processare
128 */
129 function checkPastEvent(event){
130   var promise =new Promise(function(resolve, reject){
131     setMax(event.blockNumber);                //Blocco in cui si trova
          l'evento, si controlla che sia o meno il massimo
132     if( event.returnValues[0] ===currentDomain AND
          event.returnValues[3].length >0){   //Rilevato dominio corretto
          - Verifico che sia transazione non nulla
133       var currentTransaction =event.returnValues[3];
            //Estrazione della transazione da certificare
134       var sender =event.returnValues[1];             //Estrazione del
            wallet certificatore
135       checkOldTransaction(currentTransaction).then(function (l) {
            //Controllo che la transazione non sia presente nel DB
136         if( l ===currentTransaction ){
137           checkAuthWallet(sender).then(function (s) {         //Verifica
                che il wallett sia autorizzato
138             if(s ===1) {
139               console.log('-----------------------------------------------------');
140                                             //Il server ha perso
                                                  l'evento, aggiorna il DB
141               console.log(
142               'Recupero transazione ' +currentTransaction +'...\n' +
143               'Evento di certificazione rilevato per il dominio: ' +
                    currentDomain +'\n' +
144               'Mittente: ' +sender
145               );
146               addTransaction(currentTransaction, sender).then(function (
                    _add) { //Esecuzione query di aggiornamento DB
147                 //console.log(\_add);

149                 if(_add ===currentTransaction){
150                   resolve("Transaction_exec");        //Query eseguita
                        correttamente
151                 }
152                 else
153                   resolve("Transaction_reject");         //La
                        transazione era gia' presente e non e' stata
                        inserita

155               }).catch(function (err) {
156               console.log("Error " +err);
157               reject(err);
158               });
159           }
160           else
```

```
161               resolve("Unauthorized_wallet!");

163           }).catch(function (err) {
164             console.log("Error Check Wallet " +err);
165             reject(err);
166           });

168         }
169         else
170           resolve("Already_present_TX");
171       }).catch(function (err) {
172         console.log("Error Check old_trans" +err);
173         reject(err);
174       });
175     }else{
176       //Dominio differente
177       resolve("Diff_domain");
178     }



181   });
182   return promise;
183 }

185 /**
186 * @desc Aggiornamento ultimo blocco analizzato nel DB
187 */
188 function updateMax(){
189   var sql =UPDATE server_info SET latestBlock = $max_block_tmp WHERE
        nomeServer = '$dbname';
190         con.query(sql, function (err, result) {
191           if (err)
192             throw (err);
193         });
194 }
195 /**
196 @desc Flusso principale del programma
197 */
198 function main(){
199   dbConnection().then(function(dbConnecte){    /* Connessione al DB */

201   }).then(function(recover){       /* Recupero eventi passati */
202     startServer().then(function(started){
203     updateMax();         /* Aggiornamento valore ultimo blocco
          processato */
204     listen();          /* Avvio del server in ascolto */
205   })
206   }).catch(function(err){
207     console.log("Error" +err);
208     reject(err);
209   });
```

```
211 }

213 /**
214 * @desc Funzione che si occupa lanciare il recovery degli eventi passati
215 * @returns promise
216 */
217 function startServer(){
218   var promise =new Promise( function(resolve, reject){

220       console.log('------------------Starting
              server-----------------\n');
221       currentDomain =currentDomain.trim();
222       console.log(currentDomain +"\n");
223       recovery().then(function(rec){
224         resolve("Past tx recovered");
225       }).catch(function(err){
226         console.log("Error" +err);
227         reject(err);
228       });


231   });
232   return promise;
233 }
234 /**
235 * @desc Funzione che si occupa di scaricare dallo Smart Contract
      DTTManager l'elenco degli eventi passati, a partire dall'ultimo
      blocco analizzato, contenuto nella variabile lastAnalyzedBlock
236 */
237 function recovery(){
238   var promise =new Promise(function (resolve,reject) {
239     console.log('-----------------------------------------------------');
240     contract.getPastEvents('Certification', {       //La funzione
          getPastEvents messa a disposizione dalla libreria Web3 permette
          di ottenere un oggetto contenente l'elenco degli eventi di un
          certo tipo emessi da uno specifico contratto
241       filter: {certification_domain: currentDomain},
242       fromBlock: lastAnalyzedBlock,
243       toBlock: 'latest',
244     },function(error, events){
245       console.log('Recupero transazioni...');
246       var promises =new Array();
247       //Viene utilizzato un array di promise in modo da sviluppare una
            alla volta ed evitare che vengano effettuate due insert
248       //con stessa chiave primaria (TX hash)
249       for (var i =0;i <events.length ;i++){
250         promises.push(checkPastEvent(events[i]));     //Chiamata alla
              funzione checkPastEvent(...) che verifica l'effettiva
              perdita di un evento
251       }
```

```
253     // console.log("ARRAY CHECK:", promises.length);
254     Promise.all(promises).then(function(done){
255       //console.log("check:", done);
256       //console.log("DONE!");
257       resolve("Done!");
258     },reason =>{
259       console.log(reason)
260     })

262       });

264   });
265   return promise;
266 }
267 /**
268 * @desc Il programma si mette in ascolto degli eventi emessi dallo
      SmartContract DTTManager gestendoli opportunamente
269 */
270 function listen(){
271   console.log('-----------------------------------------------------');
272   console.log('Server in ascolto...');
273     //Il server si mette in ascolto degli eventi
274     contract.events.Certification()
275     .on('data', (event) =>{
276       //console.log(event);
277       var certification_domain =event.returnValues[0];
            //Estrazione dominio di certificazione
278       if(currentDomain ===certification_domain ){       //Verifica del
            dominio
279         var currentTransaction =event.returnValues[3];
              //Estrazione della transazione da certificare
280         var sender =event.returnValues[1];          //Estrazione del
              wallet certificatore
281           checkAuthWallet(sender).then(function (s) {      //Verifico
                che il wallet sia autorizzato
282             if(s ===1) {
283               console.log('-----------------------------------------------------');
284               console.log('Evento di certificazione rilevato per il
                    dominio: ' +certification_domain +'\n' +
285               'Mittente: ' +sender +"\nTransazione: " +
                    currentTransaction ) ;

287               console.log('Server in ascolto...');
288               addTransaction(currentTransaction, sender).then(function (
                    _add) { //Esecuzione query di aggiornamento DB
289               // console.log(_add);
290                 }).catch(function (err) {
291                 console.log("INSERT error", err);
292                 });
```

```
293             //addTransaction(events[i].returnValues[3],
                    events[i].returnValues[1]);
294           }
295           else{
296             //console.log("L'indirizzo rilevato non e' autorizzato sul
                    dominio " + currentDomain + '\n\n');
297           }
298           }).catch(function (err) {
299           console.log("Error", err);

301           });
302         //var s = checkAuthWallet(sender);
303       }
304       else {
305         //console.log('Dominio errato \n\n');
306       }
307     })
308     .on('error', console.error);
309 }

311 /**
312 * @desc Settaggio iniziale dei parametri del server
313 */
314 function setServerParameters(){
315   var promise =new Promise(function (resolve,reject) {
316     con.query("SELECT * FROM server\_info WHERE nomeServer = '" +dbname
          +"'", function (err, result, fields) {
317       lastAnalyzedBlock =result[0].latestBlock;  //Estrazione
            dell'ultimo blocco analizzato fino al momento corrente
318       max_block_tmp =result[0].latestBlock;   //Variabile d'appoggio
            per aggiornamento del suddetto valore
319       if (err)
320         reject(err);
321       else if (result)
322         {
323           resolve(result.length) ;
324         }
325       });
326   });
327 return promise;
328 }

330 /**
331 @desc Configurazione connessione al DB
332 */
333 function dbConnection(){
334   var promise =new Promise( function(resolve, reject){
335     console.log('------------------------------------------------\n');
336     console.log("Connecting to DB...");
337       con =mysql.createConnection({
338       host: "localhost",
```

```
339       user: "root",
340       password: "password",
341       database: dbname
342       });
343     setServerParameters().then(function(rec){
344       console.log("Database selected\n");
345       resolve("Database OK");
346     }).catch(function(err){
347       console.log("Error" +err);
348       reject("Setting_DB_error");
349     });
350   });
351 return promise;
352 }
```

Listing 7.2 Server.js

Bibliografia

[1] La criptovaluta Bitcoin, https://bitcoin.org/it/

[2] Consoft Sistemi S.p.A. https://www.consoft.it/it/

[3] Progetto PININ, http://www.pinin-project.eu/

[4] Il consenso di Nakamoto - Christian Nyumbayire, https://www.interlogica.it/insight/il-consenso-di-nakamoto/

[5] Bitcoin: A Peer-to-Peer Electronic Cash System - Satoshi Nakamoto, https://web.archive.org/web/20161216075922/https://bitcoin.org/bitcoin.pdf/

[6] LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE, "The Byzantine Generals Problem" nel libro "ACM Transactions on Programming Languages and Systems" 1982, pp. 382-401,

[7] K. Moriarty, "PKCS #1: RSA Cryptography Specifications Version 2.2s", RFC-8017, November 2016

[8] D. Eastlake, "US Secure Hash Algorithms", RFC-6234, May 2011

[9] Trustless, Trasparenza e Scalabilità: nuovi Concetti di Valore, https://medium.com/@novaminingita/trustless-trasparenza-e-scalabilit%C3%A0-nuovi-concetti-di-valore-f7392aa96351

[10] 2009 Exchange Rate - New Liberty Standard, http://newlibertystandard.wikifoundry.com/page/2009+Exchange+Rate

[11] Bitcoin price index from July 2012 to May 2020, https://www.statista.com/statistics/326707/bitcoin-price-index/

[12] Comparsa PoS, https://bitcointalk.org/index.php?topic=27787.0

[13] VITALIK BUTERIN, "Ethereum: A Next-Generation Cryptocurrency And Decentralized Application Platform", Bitcoin Magazine, Jan 24, 2014

[14] Sito ufficiale di Ethereum, https://ethereum.org/it/

[15] Il Trilemma della Scalabilità, https://eth.wiki/sharding/Sharding-FAQs

[16] White paper di Ethereum, https://github.com/ethereum/wiki/wiki/White-Paper

[17] Price of Ethereum from January 2016 to Mayl 2020, https://www.statista.com/statistics/806453/price-of-ethereum/

[18] Documentazione di Solidity, https://solidity.readthedocs.io/en/v0.6.10/

[19] ECMA, https://www.ecma-international.org/publications/standards/Ecma-262.htm

[20] Yuan Michael Juntao, "Sviluppare applicazioni Blockchain: Guida per creare sistemi decentralizzati su reti distribuite", Apogeo, 2019, ISBN: 978-8-850-33471-1

[21] World Wide Web Consortium (W3C), https://www.w3.org/

[22] SPID, https://www.spid.gov.it/
[23] OASIS, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
[24] DID - Decentralized Identifier, https://www.w3.org/TR/did-core/
[25] Sovrin White Paper, https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf
[26] Pagin ufficiale uPort, https://www.uport.me/
[27] What is a uPort identity?, https://medium.com/uport/what-is-a-uport-identity-b790b065809c
[28] ANSA, https://www.ansa.it/
[29] Editoria: nasce ANSAcheck, notizia d'origine certificata, https://www.ansa.it/sito/notizie/politica/2020/04/06/editoria-nasce-ansacheck-notizia-dorigine-certificata_4ab2b2e7-fe4a-4785-a946-92c194e99cd8.html
[30] P. Mockapetris, “"DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION", RFC-1035, November 1987
[31] IANA, https://www.iana.org/
[32] Infura, https://infura.io/
[33] Web3, https://web3j.readthedocs.io/en/latest/
[34] Metamask, https://metamask.io/
[35] Truffle, https://www.trufflesuite.com/
[36] Ganache, https://www.trufflesuite.com/ganache
[37] Remix, https://remix.ethereum.org/
[38] Node.js, https://nodejs.org/it/
[39] DApp University, https://www.dappuniversity.com/