Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Trasferimento di Token Inter-Blockchain: attacchi e soluzioni in uno scenario di Blockchain locali.

Relatore:
Chiar.mo Prof.
Stefano Ferretti

Correlatore:
Chiar.mo Dott.
Mirko Zichichi

Presentata da:
Riccardo Mioli

Sessione I

Anno Accademico
2021-2022

Chancellor on brink of second bailout for banks
The Times, 03/Jan/2009.

Sommario

La disponibilità di connessioni a internet poco costose ed affidabili rappresenta un lusso in molti paesi in via di sviluppo e in zone rurali di nazioni sviluppate. L'utilizzo di nuove tecnologie come le blockchain risulta quindi difficile in queste aree, per quanto esse trarrebbero certamente beneficio dalla disponibilità di sistemi di pagamento digitali, decentralizzati e tolleranti ai guasti; inoltre l'uso delle blockchain in zone rurali potrebbe rendere l'abitabilità di tali aree maggiormente appetibile.

Una possibile soluzione è costituita dalle blockchain locali, ossia catene a servizio di una ristretta area geografica in cui è disponibile solamente una rete locale, da cui potrebbero ricevere vantaggio sia paesi in via di sviluppo, sia scenari industriali dove si necessiti di blockchain il cui accesso debba avvenire solamente dall'interno dell'intranet aziendale.

L'utilità delle blockchain locali risulterebbe tuttavia limitata qualora questi sistemi rimanessero totalmente isolati dal mondo esterno. Utilizzando tecnologie che permettono l'interoperabilità tra blockchain è però possibile interconnettere questi sistemi, rendendo possibile il trasferimento di asset tra diverse chain.

Le particolarità degli scenari ipotizzati lasciano però spazio ad alcune vulnerabilità, che, se sfruttate, possono condurre ad attacchi ai danni degli utenti e dell'economia delle blockchain locali. Per risolvere le problematiche individuate, sono stati quindi discussi ed implementati una serie di requisiti per la messa in sicurezza del sistema.

Indice

Introduzione

A partire dal 2008, anno in cui è avvenuta la pubblicazione del whitepaper di Bitcoin[1], il settore informatico ha visto affermarsi una nuova tecnologia: la blockchain. Il termine blockchain può essere utilizzato per indicare "un tipo di sistema e un tipo di struttura dati"[2]; nel primo caso il focus ricade principalmente sull'architettura e sul funzionamento del sistema, nel secondo sul prodotto che si origina dall'utilizzo di questa tecnologia e che è alla base del suo funzionamento[3].

La blockchain è un registro digitale distribuito reso sicuro tramite l'utilizzo della crittografia; gli utenti di tale sistema possono inviare transazioni non ripudiabili, destinate ad altri utenti o, in alcuni casi, a programmi, denominati smart contract, firmando la transazione attraverso la loro chiave privata. Le transazioni inviate dagli utenti vengono ricevute da dei nodi, che hanno il compito di conservare l'intero storico delle transazioni registrate sulla blockchain[4].

Negli anni l'interesse per le blockchain ha subito un notevole incremento, come chiaramente riscontrabile dall'enorme numero di progetti attinenti al mondo "crypto" che vengono lanciati annualmente e nella capitalizzazione del mercato delle criptovalute, di cui è visibile un grafico dell'andamento nella figura 1.1a.

[1] Vedi S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.

[2] R. Belchior et al., *A Survey on Blockchain Interoperability: Past Present, and Future Trends*, CoRR, 2021, p.4.

[3] Nel presente lavoro di tesi il termine blockchain verrà utilizzato per indicare un tipo di sistema distribuito.

[4] Spiegare approfonditamente il funzionamento della blockchain non è l'obiettivo del presente lavoro; ad ogni modo nel capitolo relativo allo stato dell'arte verranno fornite alcune informazioni aggiuntive alla base del funzionamento di tale sistema.

La sempre maggiore adozione delle blockchain comporta un incremento nell'utilizzo delle risorse di tali sistemi; avvenimenti come l'enorme innalzamento del prezzo delle transaction Fee di Maggio 2022 ne sono la prova e, per quanto le Fee associate alle transazioni degli utenti siano spesso un "male necessario" per mantenere le blockchain al sicuro da attacchi di spam di transazioni, possono rendere poco conveniente l'utilizzo di queste piattaforme per compiere transazioni tra utenti, specialmente se tali transazioni riguardano piccole somme di denaro.

Quanto descritto, unito a un numero di transazioni per secondo (TPS) limitato rispetto a circuiti delle carte di credito tradizionali (e.g. VISA[5]), ha portato alla creazione di blockchain che puntano su una maggiore velocità di esecuzione delle transazioni e basse Fee per gli utenti; alcuni esempi di blockchain con queste caratteristiche sono: Algorand, IOTA, Polygon, Polkadot, etc. Le migliori prestazioni delle nuove blockchain sono garantite mediante: 1) algoritmi di consenso differenti dalla Proof of Work; diverse scelte architetturali, di cui sono un esempio le para-chain di Polkadot; 3) assunzioni di base diverse per quanto riguarda il modello di sicurezza contro attacchi DoS, come nel caso di IOTA [6].

La nascita di nuove blockchain, che aggiungono funzionalità o migliorano aspetti di progetti già esistenti (e.g. Fee, quantitativo di TPS, etc.), ha comportato la necessità di rendere interoperabili tali sistemi.

Un esempio pratico che giustifica la necessità di blockchain interoperabili è rappresentato dalla possibilità di spostare asset digitali da una chain a un'altra: supponendo che l'utente possessore di un token ERC721 volesse vendere il proprio NFT, potrebbe spostarlo su una chain in cui tale operazione presenti costi di transazione minori in mo-

[5]Il circuito VISA è in grado di sostenere fino a 56000 transazioni per secondo, mentre, allo stato attuale, Ethereum garantisce 15 TPS. Il motivo di ciò è che blockchain come quella di Ethereum e Bitcoin sono pensate per poter essere eseguite utilizzando hardware consumer grade, dunque qualsiasi modifica al protocollo deve continuare a non richiedere hardware di alto livello, in modo tale che chiunque possa configurare e gestire un nodo. Per ulteriori informazioni si rimanda a `https://vitalik.ca/general/2021/05/23/scaling.html` e `https://en.bitcoin.it/wiki/Scalability`.

[6]In IOTA il concetto di Fee non è presente in quanto per ogni transazione che un utente vuole inviare è richiesto che esso ne validi due provenienti da altri utilizzatori del network.

do tale da incentivare i compratori a fare offerte. Le sidechain[7] possono quindi essere un modo per decongestionare blockchain pubbliche, spostando un notevole numero di transazioni su altre blockchain più veloci ed economiche.

Nel presente lavoro di tesi verranno presentate le principali tecniche che consentono di rendere le blockchain interoperabili e, dopo averne discusso gli utilizzi e i differenti tipi di implementazione, sarà proposto l'uso di tali tecniche al fine di impiegare le blockchain anche in scenari rurali, dove, a causa di scarsa o nulla connettività, l'utilizzo di blockchain come Ethereum non sarebbe praticabile. Oltre all'utilizzo in tali contesti, verrà poi proposto un uso delle blockchain locali anche in scenari industriali, o ove siano presenti blocchi alla connessione che impediscano di utilizzare blockchain tradizionali. Lo studio di soluzioni per la comunicazione tra chain in scenari che presentino limitazioni di connettività può però presentare profili di applicabilità anche in contesti "classici" nei quali l'infrastruttura utilizzata da tali sistemi vada incontro a malfunzionamenti e, pertanto, si necessiti di soluzioni temporanee per consentire comunque alle blockchain di comunicare (e.g. guasti di una backbone).

Introdotto il concetto di blockchain locale, e descritto il modello di sistema, sarà condotta un'analisi sui possibili attacchi in tali scenari. Il contributo principale di questo lavoro consiste quindi nell'individuare e descrivere tre tipologie di attacchi (fund stealing, double spending e bridge poisoning) ai danni dell'architettura proposta e testarne sei diverse esecuzioni; in seguito verranno stilati una serie di requisiti minimi e consigliati per prevenire, o quantomeno mitigare, i possibili attacchi individuati. Infine, l'ultimo contributo consiste nell'implementazione dei requisiti descritti precedentemente e nel test a livello di protocollo e di smart contract.

[7]Con il termine sidechain si fa riferimento a una blockchain da cui vengono trasferiti asset provenienti da un'altra chain; tuttavia tale espressione "implica una relazione di subordinazione che in molti casi non si può ragionevolmente dire che esista; [...] In realtà [...] (la trasferibilità) è più una proprietà degli asset che girano sulle blockchain che non delle blockchain stesse. [...] Quindi, sarebbe senza dubbio meglio usare frasi come 'la catena B può leggere la catena A', 'un relay della catena A esiste sulla catena B' o 'D è un digital asset trasferibile tra diverse chain con A come home ledger e che può anche essere usato sulla chain B' " (V. Buterin, *Chain Interoperability*, r3, 2016, p.3). Lo stesso Buterin riconosce tuttavia che il termine viene comunemente usato per indicare la trasferibilità di asset tra chain e, pertanto, verrà utilizzato anche in questo caso con tale accezione.

L'organizzazione dell'elaborato è descritta di seguito:

- il Capitolo 1 introduce lo stato dell'arte delle soluzioni che consentono l'interoperabilità tra blockchain. Verranno presentate le tre principali tecniche per trasferire dati e asset tra blockchain, commentati i pro e i contro di ogni tecnica, descritte brevemente alcune implementazioni specifiche e, infine, sarà svolta una dimostrazione pratica di trasferimento di asset tra blockchain utilizzando ChainBridge;

- il Capitolo 2 descrive il concetto di blockchain locale ed i suoi possibili utilizzi in ambito rurale e industriale. In questo capitolo, anche grazie all'utilizzo di alcune personas, sarà dettagliato il modello di sistema e il suo funzionamento;

- il Capitolo 3 presenta i possibili attacchi nei confronti del sistema; tali attacchi sono stati individuati a seguito dello studio degli smart contract di ChainBridge e di un'analisi degli attori del sistema, dei loro ruoli e dei privilegi a loro concessi. A seguito di ogni attacco saranno discusse possibili soluzioni per la mitigazione e, partendo da tali soluzioni, verranno redatti una serie di requisiti, che porranno le basi per la modifica di alcune parti del modello di sistema, degli smart contract di ChainBridge e per l'implementazione di componenti aggiuntive;

- il Capitolo 4 concerne l'implementazione dei requisiti individuati e il conseguente test. Qui saranno riportate e discusse alcune delle parti più significative relative alla fase di sviluppo, infine si descriveranno i test svolti per garantire il corretto funzionamento di quanto implementato.

- nelle conclusioni verranno discussi i risultati raggiunti, i limiti del sistema allo stato attuale e saranno forniti alcuni spunti per sviluppi futuri.

Capitolo 1

Stato dell'arte

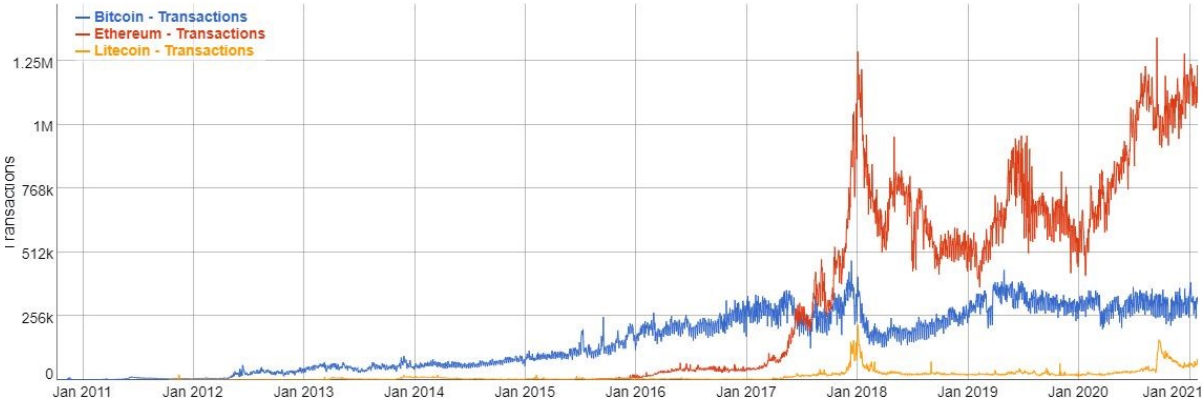
1.1 Le Blockchain

Come già anticipato nell'Introduzione, le blockchain rientrano all'interno del quadro più ampio delle Distributed Ledger Technologies (DLT), ossia tecnologie in cui una rete di nodi mantiene un registro distribuito contenente un insieme di record. Nel caso delle blockchain il contenuto del registro è rappresentati da un insieme ordinato di transazioni. Una volta che una transazione viene ricevuta da un nodo, questa viene aggregata con altre transazioni all'interno di un blocco[1]; ogni blocco, oltre alle transazioni in esso incluse, contiene ulteriori dati, tra cui un riferimento al blocco precedente. Cambiare la storia delle transazioni passate richiederebbe quindi di riscrivere anche tutti i blocchi successivi a quello modificato e, grazie a questo meccanismo, viene garantita l'immutabilità della storia delle transazioni passate.

A partire dal 2008, sono emerse svariate aree di ricerca associate alla blockchain, di cui sono un esempio: la sicurezza delle blockchain e degli smart contract, gli algoritmi di consenso alternativi alla Proof of Work (PoW), la scalabilità e interoperabilità delle blockchain, etc. Tra gli argomenti appena riportati, che sono solamente un sottoinsieme di quelli attenenti alle blockchain, gli ultimi due hanno registrato un recente

[1]Per un approfondimento sul funzionamento in generale delle blockchain si rimanda a: `https://en.wikipedia.org/wiki/Blockchain`.

(a) Capitalizzazione del mercato delle criptovalute al 2022. Immagine tratta da CoinMarketCap.



(b) Andamento del numero di transazioni giornaliere su diverse blockchain. Immagine tratta da Wikipedia.

Figura 1.1: Crescita dell'utilizzo delle blockchain.

interessamento da parte del mondo della ricerca[2]; il motivo di ciò è probabilmente legato all'aumento dell'utilizzo delle blockchain che, come visibile nella figura 1.1b, ha visto un trend crescente, arrivando, nel caso di Ethereum, a toccare le 1.25 milioni di transazioni al giorno.

Le ragioni alla base dell'incessante crescita di utilizzo delle blockchain sono molteplici: da una parte queste tecnologie hanno subito una notevole evoluzione, passando dall'essere "una pura versione peer-to-peer del denaro elettronico"[3] all'offrire la possibilità di

[2]R. Belchior et al. in *A Survey on Blockchain...*, p.2 fanno notare come il numero di paper riguardanti l'interoperabilità tra blockchain sia costantemente cresciuto a partire dal 2015, passando da 8 documenti su Google Scholar fino ai 207 del 2020.

[3]S. Nakamoto, *Bitcoin: A Peer-to-Peer...*, p.1.

eseguire codice su una virtual machine distribuita, dall'altra ciò ha aperto una serie di possibilità e mercati in precedenza inimmaginabili (e.g. Decentralized Finance (DeFi), NFT, etc.).

Ad oggi la blockchain non è più una tecnologia di "nicchia", come poteva esserlo nel 2008, e pertanto è diventato di vitale importanza poter interconnettere in modo sicuro ed efficiente sistemi nati per essere autonomi e sovrani.

1.2 Gli smart contract e il problema del Gas

Gli smart contract sono dei programmi che vengono rilasciati ed eseguiti sulla blockchain, nelle quali l'esecuzione del codice avviene in modo distribuito da parte dei minatori che operano sulla blockchain. L'esecuzione delle funzionalità offerte da uno smart contract viene richiesta dagli utenti tramite una transazione, nel cui campo `data` vengono inseriti i dati da utilizzare durante l'esecuzione del contratto; qualora le condizioni richieste per l'esecuzione della funzione venissero soddisfatte, la transazione dell'utente produrrebbe una modifica dello stato della blockchain[4].

La crescita del tasso di utilizzo delle blockchain, avvenuta negli ultimi anni, comporta un importante aspetto da tenere in considerazione, ossia l'aumento dei costi di transazione che gli utenti si trovano a dover sopportare.

Nelle blockchain la sicurezza del sistema viene mantenuta mediante l'utilizzo della crittografia e un meccanismo di incentivi e disincentivi economici: vale a dire le Fee per gli utilizzatori del network e le ricompense per i miner. In Ethereum le Fee vengono quantificate mediante il Gas, per cui il costo di una singola transazione si basa su due aspetti: le unità di Gas da utilizzare e il prezzo che si è disposti a pagare per tale quantitativo. La formula per il calcolo delle Fee è quindi:

$$TotalFee = GasUnits * GasPrice \tag{1.1}$$

Il quantitativo di Gas utilizzato dipende dalla lunghezza e complessità dell'operazione che il soggetto inviante la transazione vuole eseguire; ad esempio, un semplice invio di

[4]Per ulteriori informazioni sugli smart contract si rimanda a P. De Filippi et al. *Smart Contracts*, Internet Policy Review, Vol 10(2), 2021.

Ethereum richiede 21.000 unità di Gas, mentre una chiamata a una funzione di uno smart contract può arrivare a richiedere centinaia di migliaia di unità di Gas se le operazioni che vengono svolte riguardano procedure costose come la scrittura di valori nello storage dello smart contract. Il Gas può quindi essere visto come "potenza computazionale" della EVM che gli utenti acquistano per eseguire le loro operazioni.

Il prezzo del Gas viene espresso in GWei[5] e dipende dalla congestione del network: in momenti di alto utilizzo della blockchain il costo di una transazione aumenta, in quanto tutti i partecipanti desiderano che le loro transazioni vengano eseguite, ma il Gas utilizzabile per ogni blocco è limitato; al contrario, in momenti in cui il network non viene utilizzato al massimo delle sue capacità, il costo del Gas sarà inferiore poichè la richiesta di utilizzo è bassa[6].

Spiegato il funzionamento del Gas, vediamo ora alcune implicazioni. Il 1 Maggio 2022, a seguito del lancio dell'NFT Otherside, prodotto dagli stessi autori del Bored Ape Yacht Club NFT, la blockchain di Ethereum ha visto un impennata del prezzo del Gas, che è passato da poche decine di GWei a oltre 5000 GWei a causa dell'alta richiesta per l'NFT. Effettuare una chiamata a uno smart contract, ad esempio per concludere una compravendita di un NFT, poteva quindi comportare il pagamento di Fee pari a 1 Ether, anche per transazioni di valore decisamente inferiore[7].

Sotto il punto di vista appena descritto potrebbe sembrare che il Gas comporti solo svantaggi per gli utenti di Ethereum; tuttavia, come accennato in precedenza, la funzione delle Fee è quella di disincentivare comportamenti malevoli sulle blockchain. Infatti un

[5]Un GWei corrisponde a $1 * 10^{-9}$ Ethereum, ossia a 0.000000001 Ether.

[6]A partire da Agosto 2021, ossia dal lancio del cosiddetto London Upgrade, il modello delle fee di Ethereum è leggermente cambiato, in quanto il Gas Price è ora composto da due componenti: la Base Fee e la Tip Fee. La Base Fee è una sorta di prezzo di riserva stabilito automaticamente da parte del network sulla base dell'utilizzo medio della piattaforma; la Tip Fee viene invece assegnata dagli utenti per incentivare i miner a includere nel prossimo blocco la propria transazione al posto di quelle degli altri utenti. La Base Fee, a differenza della Tip Fee, viene bruciata, creando, in periodi di alto utilizzo del network, una deflazione della valuta.

[7]Quanto descritto è avvenuto per alcune compravendite su OpenSea nelle quali il costo delle Fee ha superato di gran lunga quello dell'NFT acquistato. Per ulteriori informazioni si rimanda a: `https://mashable.com/article/ethereum-gas-fees-skyrocket-bored-ape-yacht-club-otherside-nft-launch`.

soggetto che volesse bloccare il network inviando continuamente transazioni per impedire che altri utenti possano a loro volta effettuare transazioni si troverebbe a dover spendere un enorme somma di denaro in Gas e dunque l'attacco, che costituisce in tutto e per tutto una forma di Denial of Service (DoS), risulterebbe economicamente non conveniente. La seconda funzione del Gas è quella di incentivare i miner a continuare ad aggiungere i blocchi contenenti le transazioni; infatti, oltre ai nuovi Ether che vengono creati per ogni blocco minato, i miner possono tenere parte delle Fee come ulteriore ricompensa.

Per quanto le Fee consentano di impedire attacchi alle blockchain, risulta evidente come il loro pagamento possa scoraggiare lo sviluppo e il rilascio di smart contract sulle blockchain a causa dei costi per interagirvi; per questo motivo negli ultimi anni vi è stata una notevole proliferazione di chain che propongono differenti soluzioni a livello di protocollo ed architettura al fine di consentire costi di transazione più bassi rispetto a piattaforme come Ethereum e Bitcoin. Sebbene ciò ricada a beneficio degli utilizzatori delle blockchain, questi si trovano ora ad operare in uno spazio frammentato e a possedere token di varia natura su chain differenti.

1.3 I token come asset e il loro trasferimento inter-blockchain

Nonostante la definizione di token sia in continua evoluzione "esiste un accordo generale che i 'token' siano 'crypto-asset'; questi ultimi sono definiti in senso lato come 'una rappresentazione digitale crittograficamente protetta di valore o di diritti contrattuali che utilizza un qualche tipo di DLT e che può essere trasferita, memorizzata o scambiata elettronicamente."[8].

M. Zichichi et al. stilano una tassonomia dell'attuale panorama dei token: nello specifico viene effettuata una distinzione tra token non fungibili, ossia contraddistinti da un elemento di unicità, fungibili, ossia identici tra loro o "funzionalmente equivalenti"[9], e ibridi. Gli elementi compositivi dei token possono poi garantire "comportamenti"

[8]M. Zichichi et al., *MOATcoin: Exploring Challenges and Legal Implications of Smart Contracts Through a Gamelike DApp Experiment*, ACM, 2020, p.5.

[9]Ibidem.

differenti che possono rendere i token: trasferibili o non trasferibili, divisibili o indivisibili, distruttibili o indistruttibili, mintable o meno, etc.[10]

Un ulteriore aspetto dei crypto-asset è che questi possono essere nativi o non nativi: nel primo caso dispongono di una propria blockchain, nel secondo sono "asset costruiti sopra un'altra blockchain"[11]. In Ethereum l'implementazione di token non nativi viene svolta sulla base di alcuni standard, di cui i principali esempi sono l'ERC-20 e l'ERC-721; in entrambi i casi gli standard prevedono alcune funzionalità di base, come ad esempio la trasferibilità dei token, che possono poi essere estese a seconda delle necessità del progetto per cui il token sta venendo realizzato.

I token che fanno uso dello standard ERC-20[12] rientrano all'interno della famiglia dei token fungibili. Attraverso questi token è possibile implementare sistemi di scambio, o pagamento, che usano valute alternative a quella nativa della blockchain. I token ERC-20 possono però anche avere impieghi differenti rispetto a un normale sistema di pagamento, ad esempio sistemi di voto che utilizzano il quantitativo di token a disposizione di un utente per assegnare un peso al suo voto: in questi casi si parla di utility token. Infine, i token ERC-20 possono anche rappresentare una forma di investimento (i.e. utility token) ed infatti sono spesso utilizzati nella raccolta di capitali. Le suddivisioni appena presentate sono spesso sfumate e sovrapponibili; in effetti non è inusuale che un token ERC-20 possa svolgere funzionalità di utility e security token.

L'ERC-721 è lo standard impiegato dai Non-Fungible Token[13]. Come già accennato, questo tipo di token si differenzia dai token fungibili a causa dell'unicità di ogni token rispetto agli altri e la differenza tra i token è generalmente rappresentata mediante un id univoco.

A causa delle loro caratteristiche i Token Non Fungibili sono particolarmente utilizzati in campo artistico e collezionistico. Tuttavia è possibile impiegarli in qualsiasi ambito sia richiesto di identificare univocamente un prodotto: un esempio di possibile utilizzo può essere nell'ambito della biglietteria digitale, ma anche in campo videoludico, etc.

Per quanto l'unicità degli NFT li renda particolarmente adatti come security token,

[10] Ibidem.

[11] Ibidem.

[12] Lo standard è disponibile presso: `https://eips.ethereum.org/EIPS/eip-20`.

[13] Per ulteriori informazioni si rimanda a: `https://eips.ethereum.org/EIPS/eip-721`.

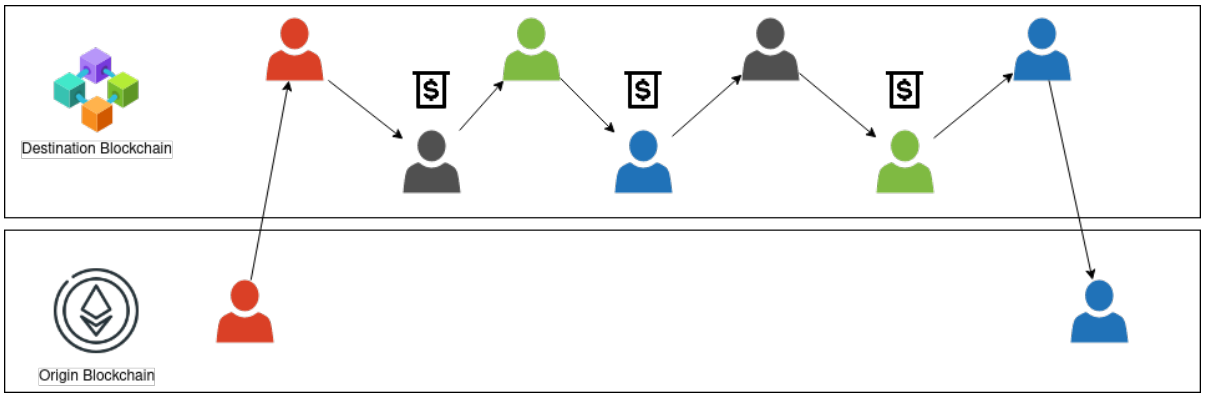


Figura 1.2: Trasferimento asset interblockchain.

questi token possono essere utilizzati anche come "chiave" di accesso a servizi o benefit offerti ai loro possessori[14], pertanto possono assumere anche funzionalità tipiche degli utility token.

La presenza di molteplici blockchain consente di utilizzare i propri token su diversi network, che possono offrire caratteristiche differenti a livello di user base, Fee di utilizzo, velocità di produzione dei blocchi, sicurezza, etc.; la necessità di spostare i propri token per usufruire di tali caratteristiche è quindi nata dalla proliferazione di nuove blockchain che si differenziano tra loro.

Presentiamo ora un esempio pratico che consentirà di comprendere il motivo per cui un individuo potrebbe voler trasferire i propri asset tra chain differenti: supponiamo che un utente disponga di un NFT su una blockchain come Ethereum e che voglia mettere all'asta il suo token. In periodi di alta congestione del network, l'asta potrebbe essere poco partecipata a causa delle alte Fee per l'invio di una transazione; il venditore potrebbe quindi spostare il proprio token su una blockchain differente per consentire agli offerenti di effettuare molteplici proposte su un network il cui utilizzo è poco costoso. Terminata l'asta, ulteriori compravendite potrebbero svolgersi sul network non congestionato per poi migrare il token sulla blockchain di origine qualora l'ultimo compratore volesse tenere il token per un tempo indefinitamente lungo.

[14]Ne sono un esempio i CryptoBaristas, che garantiscono sconti sull'acquisto di prodotti ai proprietari dei token. Per ulteriori informazioni si rimanda al sito web del progetto: `https://cryptobaristas.com/holder-perks/`.

L'ultimo spostamento del token non è obbligatorio; tuttavia l'utente che possiede il token potrebbe considerare una blockchain più sicura di un'altra, in quanto presenti un maggior numero di nodi, o per una preferenza relativa all'algoritmo di consenso distribuito utilizzato. Nelle blockchain private o permissioned, ad esempio, la sicurezza del network si basa sulla reputazione dei nodi che la compongono. Tali chain, avendo un numero di utilizzatori ristretto, hanno costi di transazione irrisori; tuttavia un utilizzatore potrebbe preferire muovere alcuni asset al di fuori della catena privata in quanto ritiene che una blockchain pubblica, in cui l'elemento della fiducia nei nodi non è presente, possa garantire una sicurezza maggiore per i suoi asset.

Un esempio della procedura appena descritta è rappresentato dalla figura 1.2.

L'asset bridging, ossia lo spostamento di token tra blockchain, viene reso possibile da soluzioni che permettono l'interoperabilità tra blockchain. Tuttavia nuove opportunità nascono dalla possibilità di far comunicare blockchain differenti; un esempio è rappresentato da EthereumLottery.io, che utilizza uno smart contract sulla blockchain di Ethereum per la gestione di una lotteria, ma sfrutta gli header dei blocchi di Bitcoin come sorgente di randomicità[15].

Un ulteriore caso d'uso dell'interoperabilità tra chain è rappresentata dallo state pinning: tale pratica consiste nel pubblicare su una blockchain, diversa da quella che si sta utilizzando, alcune informazioni sullo stato della piattaforma (e.g. hash dell'ultimo blocco) in modo che in seguito sia possibile verificare che non vi siano state modifiche. Un esempio può essere quello di una blockchain "pay-per-use" che viene messa in pausa per un periodo di tempo indefinito per poi essere riattivata su richiesta: gli utenti del servizio, per garantire che il prestatore di servizi non effettui modifiche arbitrarie allo stato della catena, possono effettuare il pin dell'hash dell'ultimo blocco su una chain pubblica, come quella di Ethereum. In questo modo è possibile avere la garanzia che qualsiasi modifica da parte del prestatore di servizi sia facilmente individuabile[16].

[15] V. Buterin, *Chain Interoperability*, p.6.

[16] L'idea alla base dello state pinning è simile a quella della cosiddetta marca temporale.

1.4 Tecniche per l'interoperabilità tra blockchain

R. Belchior et al., adottando l'European Interoperability Framework model, evidenziano come l'interoperabilità tra chain possa essere considerata su quattro livelli[17]:

1. l'interoperabilità tecnica riguarda i formati, i protocolli di comunicazione, le interfacce e, più in generale, tutti quei meccanismi che consentono il passaggio di dati da un sistema a un altro;

2. l'interoperabilità semantica riguarda come i dati vengono interpretati a partire da una specifica ontologia, ossia da un modello di informazione. Questo tipo di interoperabilità, a differenza dell'interoperabilità tecnica, riguarda quindi l'interpretazione che viene fatta di un dato da parte di sistemi differenti e non i meccanismi per la sua trasmissione;

3. l'interoperabilità organizzativa comprende gli aspetti che concernono i processi di business di diverse organizzazioni e la loro integrazione;

4. l'interoperabilità legale "assicura che le organizzazioni possano cooperare sotto 'differenti framework legali, policy e strategie'."[18].

Gli ultimi due tipi di interoperabilità richiedono l'interazione tra organizzazioni differenti, che abbiano processi e framework consolidati. Nell'ambito dell'interoperabilità tra blockchain questa necessità non sembra ancora essersi manifestata, probabilmente anche per una mancanza di un massiccio utilizzo di questa tecnologia a livello industriale, che, dove presente, allo stato attuale assume più i connotati di una sperimentazione[19]; dunque la tendenza è quella a considerare solamente l'interoperabilità tecnica e semantica dei sistemi.

[17]Si veda R. Belchior et al., *Do You Need a Distributed Ledger Technology Interoperability Solution?*, TechRxiv, 2022, p.11.

[18]Ibidem.

[19]Ne sono un esempio l'utilizzo degli NFT da parte di brand come Alfa Romeo. Per ulteriori informazioni si rimanda a `https://www.ilsole24ore.com/art/alfa-romeo-tonale-ecco-perche-nft-aumentano-valore-residuo-AEXPjBEB`.

Stabiliti i possibili tipi di interoperabilità esistenti, valutiamo ora alcuni aspetti da tenere in considerazione nel momento in cui si valutano soluzioni per consentire l'interoperabilità tra blockchain.

Una prima caratteristica da considerare è se l'interoperabilità debba avvenire a livello verticale o orizzontale[20]. Nel primo caso vengono interconnessi subnetwork appartenenti alla stessa blockchain, nel secondo vengono fatti comunicare network differenti[21]. La seconda caratteristica è la modalità di interoperabilità che si vuole ottenere, ciò dipende dal tipo di contenuto che si necessita trasmettere da una chain all'altra. Le modalità sono due: dati o asset[22]; nel primo caso vengono scambiati dati che verranno processati sulla chain di destinazione, nel secondo vengono inviati direttamente token. Lo scambio diretto di asset può essere considerato più limitato rispetto alla modalità che riguarda i dati; infatti inviando dati da una blockchain a un altra possono essere implementate logiche per la conversione di questi in asset[23]. Infine, nel momento in cui viene scelta una soluzione per consentire l'interoperabilità tra blockchain, è necessario valutare se essa richieda o meno modifiche alla piattaforma. Qualora venissero richieste modifiche, esse diventano parte integrante dell'architettura della piattaforma, in caso contrario l'interoperabilità è raggiunta mediante "contratti sulla blockchain o server esterni alla blockchain"[24], dunque a livello "applicativo".

Ad oggi "non ci sono standard *de facto* per l'interoperabilità e l'architettura delle blockchain"[25] e la soluzione migliore da adottare dipende fortemente dai requisiti che un progetto si trova a dover soddisfare e dal caso d'uso della piattaforma; tuttavia, nel variegato panorama delle soluzioni per l'interconnessione di blockchain[26] emergono tre principali tecniche: Hash Time Lock Contracts, Relay e Notary Schemes.

[20]Cfr. R. Belchior et al., *Do You Need...*, p.8.

[21]È il caso del bridge tra Ethereum e il Polygon Network.

[22]Ivi p.15.

[23]La tecnica appena descritta è quella utilizzata da ChainBridge.

[24]P. Robinson, *Survey of Crosschain Communications Protocols*, CoRR, 2021, p.4.

[25]R. Belchior et al., *A survey on Blockchain...*, p.4.

[26]Per una disamina sui protocolli e sulle soluzioni esistenti si rimanda a S. Johnson et al., *Sidechains and interoperability*, CoRR, 2019 e P. Robinson, *Survey on Crosschain Communications Protocols*, CoRR, 2021.

1.4.1 Hash Time Lock Contracts

Gli Hash Time Lock Contracts (HTLC) sono la più semplice tra le tecniche che permettono l'interoperabilità tra blockchain, la loro semplicità li rende tuttavia adatti solamente allo scambio di asset su chain differenti, ossia i cosiddetti atomic swap.

Il funzionamento di tale sistema si basa sulla presenza di due contratti, uno per ogni blockchain interessata dallo swap di token, o valute, e sulla presenza di due agenti, che per semplicità assumeremo essere due individui reali, ma che potrebbero essere organizzazioni, bot, etc. I due agenti, che in questo caso chiameremo Alice e Bob, concordano off-chain uno swap di asset; Alice effettua quindi il deploy sulle chain dei contratti che verranno utlizzati per svolgere lo scambio. L'utilizzo di smart contract consente di rimuovere l'elemento della fiducia tra i due utenti, in quanto gli asset di cui verrà effettuato lo scambio verranno depositati presso gli smart contract.

Una volta pubblicati su blockchain gli smart contract, questi vengono configurati inserendo un committment value e un limite temporale per completare lo scambio; in questo modo, qualora una delle due parti decidesse di non procedere con lo scambio, i fondi non rimarranno bloccati nei contratti, ma saranno liberamente prelevabili dal proprietario. Il commitment value consiste nell'hash H di un numero R, scelto da Alice, e svolge il ruolo di segreto condiviso tra le due parti per evitare che altri individui possano prelevare gli asset depositati nei contratti, lo scambio può essere infatti portato a termine solamente fornendo il valore R che ha generato l'hash H.

Terminato il setup dei contratti, Alice deposita il proprio asset e Bob, una volta verificato che gli asset depositati da Alice corrispondano a quanto concordato, effettua lo stesso procedimento. Alice si reca quindi sulla blockchain di Bob, inserisce il valore R che ha generato l'hash H e preleva l'asset; Bob, visionato il valore R inserito da Alice, si reca sulla blockchain di Alice, inserisce lo stesso valore nello smart contract e preleva il token depositato da Alice.

Nell'immagine 1.3 è rappresentato il processo appena descritto.

La soluzione presentata non è esente da possibilità di comportamenti scorretti, nello specifico le parti potrebbero depositare i propri fondi, ma Alice, vedendo un deprezzamento del token di Bob, potrebbe decidere di non rivelare il valore R, lasciando bloccati gli asset di entrambi i partecipanti fino allo scadere del tempo. Per risolvere questa

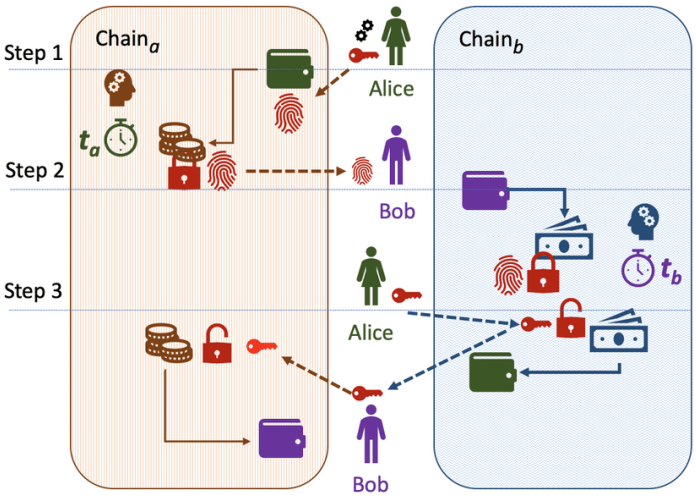


Figura 1.3: Esempio funzionamento degli Hash Time Lock Contracts. Immagine tratta da J. Xu et al., *A Game-Theoretic Analysis of Cross-Chain Atomic Swaps with HTLCs*, 2021, p.2.

problematica, una delle soluzioni proposte è quella di aggiungere agli asset depositati un collaterale che funga da indennizzo qualora una delle due parti non dovesse partecipare correttamente allo scambio.

Come accennato in precedenza, questa soluzione consente di effettuare scambi atomici tra asset su blockchain differenti; tuttavia la semplicità degli Hash Time Lock Contracts consente solamente di operare in modalità asset e non permette di scambiare dati tra blockchain. Un ulteriore aspetto da considerare è che, dovendo essere raggiunto un accordo off-chain tra le parti, può essere complicato automatizzare l'intero processo. Infine, l'approccio appena presentato è ulteriormente limitante in quanto, avvenendo uno scambio di token tra parti, non è possibile per un utente muovere i suoi asset "asimmetricamente" da una chain a un'altra.

1.4.2 Relays

I Relay rappresentano una soluzione più flessibile rispetto agli Hash Time Lock Contracts. Tramite questi sistemi è possibile replicare "lo stato di una blockchain sorgente su una blockchain di destinazione e in questo modo consentire alla blockchain di desti-

nazione di verificare l'esistenza di alcuni pezzi di stato della blockchain di origine"[27], consentendo di fatto di trasmettere dati o asset a un'altra chain senza che siano richieste due parti che raggiungono un accordo off-chain. Il funzionamento dei Relay si basa sugli header dei blocchi e sulla verifica delle transazioni che sono in essi contenuti.

L'architettura di un Relay è tipicamente la seguente: sulla blockchain che si vuole rendere interoperabile è presente uno smart contract in grado di memorizzare gli header di blocchi e verificare che essi siano compatibili con l'algoritmo di consenso utilizzato dalla chain di origine. Il Relay "è fatto funzionare da client off-chain che inviano continuamente gli header dei blocchi da una blockchain sorgente ad una di destinazione. La blockchain sorgente essenzialmente viene replicata nella blockchain di destinazione"[28]. Qualsiasi utente può quindi inviare l'header di un blocco e guadagnare un credito che potrà riscuotere in seguito; la procedura di invio di un blocco non è limitata in alcun modo in quanto la sicurezza è garantita dalla verifica degli header ad opera dello smart contract del Relay, che segue l'algoritmo di consenso della blockchain di provenienza.

Per trasmettere dati da una blockchain a un'altra, un individuo invia al contratto Relay l'hash del blocco in cui è presente la sua transazione, i dati raw inerenti alla transazione, una merkle proof per la transazione e una Fee per utilizzo del servizio, necessaria a coprire i costi dell'invio dell'header del blocco. Ricevuti i dati, il contratto Relay utilizza la merkle proof dell'utente per verificare che la transazione sia effettivamente compatibile con il block header. Qualora la verifica della transazione andasse a buon fine, verrebbe chiamata una funzione per il processing dei dati inviati dall'utente, con i quali saranno attuate operazioni differenti a seconda della finalità del creatore del contratto (e.g. mint di token, ulteriori chiamate di funzione, state pinning, etc.).

Nella figura 1.4 è possibile visionare il procedimento appena descritto.

Una delle problematiche dell'utilizzo dei Relay è il costo associato al salvataggio dell'header dei blocchi nello storage dello smart contract che effettua le verifiche: i costi associati all'invio dei dati possono infatti non essere coperti dalle Fee per l'utilizzo del servizio versate dagli utenti che vogliono trasmettere le proprie transazioni. In effetti, come evidenzia P. Robinson[29], questo è esattamente quanto accaduto a BTCRelay, uno

[27] P. Frauenthaler et al., *Leveraging Blockchain Relays for Cross-Chain Token Transfers*, 2020, p.1.
[28] Ibidem.
[29] Si veda P. Robinson, *Survey of Crosschain...*, p.6.

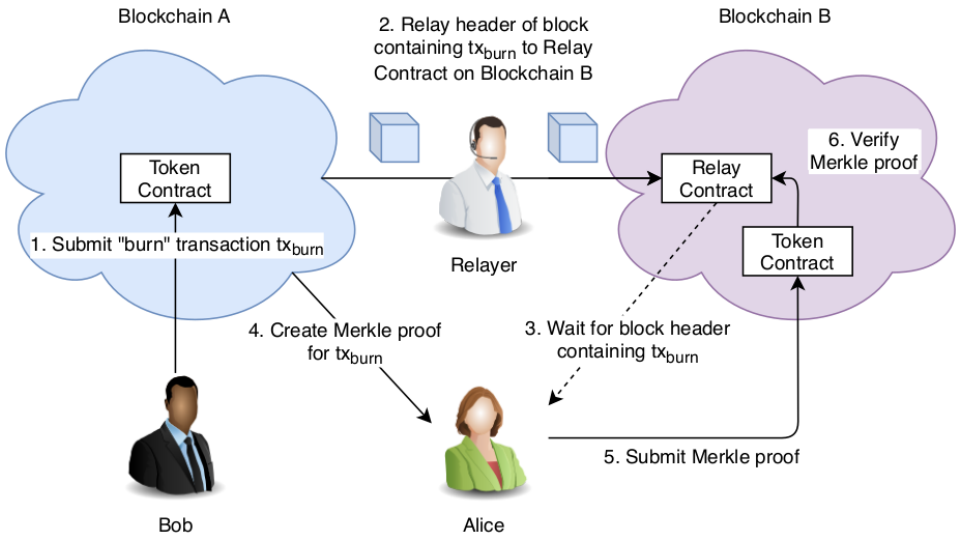


Figura 1.4: Funzionamento di un relay. Immagine tratta da P. Frauenthaler et al., *Leveraging Blockchain Relays...*, p.4.

dei primi Relay ad interconnettere la blockchain di Ethereum con quella di Bitcoin. Il problema dei costi può essere ulteriormente esacerbato dall'utilizzo di algoritmi di consenso come il Proof of Stake (PoS); tali algoritmi utilizzano infatti un set di validatori che cambia ciclicamente ad ogni epoca, memorizzare l'header di un blocco e le chiavi pubbliche di tutti i validatori dell'epoca rende l'operazione ancora più costosa, specie se il numero di validatori è alto. Per mitigare il problema M. Diez et al. propongono di inviare le chiavi pubbliche dei validatori insieme ad ogni header di blocco, in questo modo le chiavi vengono utilizzate per verificare gli header senza però che vengano salvate nello storage del contratto, comportando di fatto un notevole risparmio in termini di Gas[30].

Per quanto siano possibili ottimizzazioni a livello di utilizzo del Gas, la principale limitazione dei Relay è che questi richiedono l'invio continuo degli header dei blocchi, anche se nessun utente dovesse richiedere di verificare una propria transazione. P. Frauenthaler et al. effettuano una stima dei costi d'esercizio della loro soluzione[31], arrivando a stimare un costo annuo di 1.160.495,47 euro per l'invio di 4 header al minuto con un costo del Gas pari a 10 GWei e un valore di Ethereum pari a 204,79 EUR/ETH. Considerando le

[30]Cfr. M. Diez et al., *Verilay: A Verifiable Proof of Stake Chain Relay*, CoRR, 2022, p.7.
[31]Si veda P. Frauenthaler et al., *Leveragin Blockchain Relays...*, p.3.

fluttuazioni del mercato delle criptovalute e il costo del Gas, la pubblicazione costante degli header dei blocchi, ricompensata mediante Fee per la verifica delle transazioni, non rappresenta un modello sostenibile, soprattutto, come dimostrato dal caso di BTCRelay, qualora la base di utenti che utilizzano il Relay fosse piccola.

1.4.3 Notary Schemes

A differenza delle due tecniche presentate precedentemente, i Notary Schemes prevedono la presenza di terze parti fidate, che monitorano determinati contratti su una blockchain di origine e attendono che un determinato evento si manifesti. Nel momento in cui viene generato l'evento, le parti fidate trasmettono i dati ad esso relativi sulla blockchain di destinazione.

Per mitigare problematiche relative alla fiducia negli osservatori, i Notary Schemes prevedono solitamente un sistema di voto; in questo modo è possibile trasmettere dati alla blockchain di destinazione solamente qualora la maggioranza degli osservatori concordasse che un evento è realmente avvenuto.

Pur richiedendo un elemento di fiducia, che nelle soluzioni presentate precedentemente non è presente, i Notary Schemes consentono di trasmettere tra blockchain sia dati sia asset e, a differenza dei Relay, non richiedono l'invio continuo di dati tra una blockchain e un'altra, consentendo un risparmio notevole agli utenti del sistema, che sono quindi incentivati ad utilizzare questo sistema.

Nella tabella 1.4.3 è riportato un confronto tra le caratteristiche delle soluzioni descritte nelle sezioni precedenti.

Tecnica	**Modalità di trasferimento**	**Pro e Contro**

HTLCs	• Asset	• Nessun intermediario ed elemento di fiducia richiesto. • Necessari due utenti che raggiungano un accordo off-chain, difficile automazione. • Non è possibile trasmettere dati. • Non è possibile trasmettere asset "asimmetricamente".
Relays	• Asset • Dati	• Nessun intermediario ed elemento di fiducia richiesto. • Costi per l'invio degli header dei blocchi potenzialmente alti.
Notary Schemes	• Asset • Dati	• Bassi costi di utilizzo. • È presente un elemento di fiducia, rappresentato dagli osservatori della blockchain.

Capitolo 2

Blockchain locali

Presentato lo stato dell'arte delle tecniche che consentono di rendere interoperabili due o più blockchain, discutiamo ora l'utilizzo di tali soluzioni in un contesto di blockchain locali e le opportunità offerte da questo tipo di chain.

2.1 Motivazione e casi d'uso

Definiamo blockchain locale una blockchain pubblica[1], che è però utilizzata da un ristretto numero di persone all'interno di uno specifico network. La situazione per cui l'utilizzo di una blockchain avviene solo da parte di persone all'interno di uno stesso network può dipendere da svariati motivi; in questa sede assumiamo che ciò avvenga a causa di limitazioni nella connettività, che a loro volta possono derivare da fattori che vanno dalla propagazione limitata del segnale, ad esempio a causa di aspetti geografici del territorio, a blocchi nell'accesso a determinate risorse derivanti dalla segmentazione della rete per ragioni di sicurezza. Nel primo caso il confine del network è legato ad aspetti fisici del territorio, nel secondo la barriera è digitale (e.g. firewall, VLAN, etc.). La necessità di utilizzare una blockchain locale può originarsi anche a causa di guasti all'infrastruttura di rete che copre una vasta zona, ad esempio qualora una o più backbone dovessero subire una interruzione. Quanto descritto potrebbe avvenire in caso di un'azione di

[1]Con blockchain pubblica si intende una chain in cui ogni nodo è libero di leggere e scrivere sul registro distribuito di cui fa parte.

sabotaggio in scenari di guerra e avrebbe l'effetto di impedire l'accesso a internet a un largo numero di persone, che risulterebbero impossibilitate a interagire con servizi e blockchain tradizionali.

Ad oggi è presente un ristretto numero di studi che prendano in considerazione la possibilità di utilizzare le blockchain in scenari in cui la connettività è totalmente assente, instabile, o, più in generale, in cui sono presenti forti limitazioni nell'utilizzo di una connessione internet (i.e. impossibilità di contattare determinati indirizzi); tuttavia, come vedremo, l'utilizzo di blockchain locali potrebbe risultare particolarmente utile in contesti del genere.

2.1.1 Scenario rurale

Al momento della scrittura di questa tesi, la ricerca sembra essersi orientata all'utilizzo di blockchain in scenari di scarsa connettività mediante l'introduzione di middleware che sfruttino tecnologie come gli SMS[2] per favorire la comunicazione tra utenti e blockchain. Tuttavia, tali soluzioni potrebbero non sempre essere impiegabili; infatti, come emerge dal report della International Telecommunication Union[3], seppur la copertura telefonica e internet sia aumentata durante il periodo pandemico, "circa 2.9 miliardi di persone rimangono offline, il 96% delle quali vive in paesi in via di sviluppo"[4]. Il gap tecnologico tra paesi sviluppati e in via di sviluppo, pur essendosi ridotto negli ultimi anni, rimane comunque evidente; infatti "è doppiamente più probabile che le persone nelle aree urbane usino internet rispetto a quelle nelle aree rurali"[5]. Inoltre, in zone del mondo poco sviluppate, grandi aree rimangono senza connettività a banda larga, ad esempio, "in Africa, il 18% della popolazione rurale non ha alcuna copertura di rete mobile e un altro 11% ha solo la copertura 2G. Ciò significa che quasi il 30% della popolazione rurale non può accedere a Internet. Il divario di copertura è quasi altrettanto significativo nelle

[2]Alcuni esempi sono Y. Lin et al., *BcMon: Blockchain Middleware for Offline Networks*, arXiv, 2022 e D. Maldonado-Ruiz et al., *Secure and Internet-Less Connectivity to a Blockchain Network for Limited Connectivity Bank Users*, 2020.

[3]Si veda International Telecommunication Union, *Measuring digital developement. Facts and Figures*, ITUPublications, 2021.

[4]Ivi p.1.

[5]Ivi p.6.

Americhe, dove il 22% della popolazione rurale non è affatto coperto e un altro 4% è coperto solo dal 2G"[6].

Un ulteriore aspetto da considerare è quello del reddito degli utilizzatori dei servizi di rete; infatti, anche dove un qualche genere di connessione fosse disponibile, in diverse aree del mondo avere un piano dati può rappresentare costi proibitivi per un individuo con scarso reddito[7]. Soluzioni che sfruttino l'internet satellitare per consentire l'utilizzo delle blockchain o piattaforme specificatamente ideate per questo scopo, come Blockstream[8], sono quindi difficilmente impiegabili, considerando i costi per l'installazione dell'infrastruttura necessaria e i contratti di utilizzo.

Avere a disposizione una connessione veloce e affidabile è spesso una condizione necessaria per poter utilizzare tecnologie moderne e, come abbiamo visto, tale requisito non è di facile soddisfacibilità in diverse zone del mondo in via di sviluppo; tuttavia, problematiche simili sono comuni anche ad aree periferiche di paesi sviluppati, di cui un esempio è certamente rappresentato dai piccoli nuclei abitativi in zone montuose o dai rifugi di montagna.

Le problematiche che si incontrano nei contesti appena descritti hanno un forte impatto sull'utilizzabilità della blockchain. Infatti, non poter disporre di una connessione a internet affidabile comporta disagi qualora si volesse partecipare attivamente all'infrastruttura della blockchain inizializzando un nodo, operazione che può richiedere la ricezione di centinaia di gigabyte di dati, ma anche nella sincronizzazione degli ultimi blocchi processati dai miner, per la quale è richiesta una connessione stabile e che non subisca lunghe interruzioni, o più semplicemente nell'utilizzare wallet per inviare le proprie transazioni a un nodo.

Alla luce di quanto visto, l'utilizzo di blockchain in scenari rurali sembrerebbe di difficile realizzabilità; tuttavia tale tecnologia, per ora a vantaggio di aree maggiormente sviluppate del mondo, può essere di enorme utilità anche in zone in via di sviluppo o ove semplicemente non ci sia una connessione alla WAN (Wide Area Network). Alcuni utilizzi di cui beneficerebbero tali aree possono essere:

[6]Ivi. p.12.

[7]International Telecommunication Union, *Measuring digital ...*, p.15.

[8]Per ulteriori informazioni si rimanda al sito web della piattaforma: `https://blockstream.com/`.

- attirare investimenti in paesi in via di sviluppo, garantendo la tracciabilità e verificabilità delle transazioni;
- rendere attraenti per le nuove generazioni zone del mondo rurali mediante l'utilizzo di tecnologie moderne[9];
- offrire servizi legati al territorio in modo da rendere desiderabile visitare o abitare una determinata zona, per quanto questa possa essere isolata;
- favorire l'utilizzo di nuove tecnologie[10], rendendo competitive aree del mondo emergenti;
- avere un sistema che consenta di effettuare pagamenti elettronici e garantisca i diritti di proprietà degli individui mediante la crittografia anche in zone in cui vi è instabilità politica e una infrastruttura di rete inadeguata.

Per quanto gli obiettivi appena descritti possano sembrare relegati a una ristretta area del mondo, osservando la figura 2.1 è possibile rendersi conto come molte delle zone in cui le infrastrutture di rete sono inadeguate siano anche quelle in cui sono in corso guerre, che quindi trarrebbero beneficio dall'utilizzo di sistemi non centralizzati e, pertanto, tolleranti ai guasti e difficilmente censurabili; inoltre, come già accennato in precedenza, anche zone montuose di paesi sviluppati possono condividere problematiche simili a quelle di zone del mondo in via di sviluppo (e.g. scarsa appettibilità nell'abitarvi, difficoltà nell'utilizzo di servizi di pagamento digitali, etc.).

Le possibilità descritte possono essere soddisfatte mediante l'adozione di una blockchain locale, utilizzabile dagli individui che ricadono sotto una stessa LAN (Local Area

[9]Un esempio sotto questo punto di vista è CityDAO, un progetto di città in una area rurale del Wyoming in cui i partecipanti vogliono applicare idee economiche non convenzionali e prendere decisioni collettive. Per ulteriori informazioni si rimanda a `https://vitalik.ca/general/2021/10/31/cities.html`.

[10]E. Barinaga in *Sarafu: A cryptocurrency for kenyan rural communities*, SAGE Business Cases, 2022, descrive l'introduzione di una criptovaluta come mezzo di pagamento digitale in Kenya. Al momento il progetto è ancora in una fase di sperimentazione e stanno venendo studiate alcune possibili soluzioni per incentivare l'utilizzo della valuta; tuttavia le premesse sembrano essere positive e gli autori segnalano che a Giugno 2019 il 90% della popolazione della zona interessata dallo studio effettuava scambi sulla piattaforma.

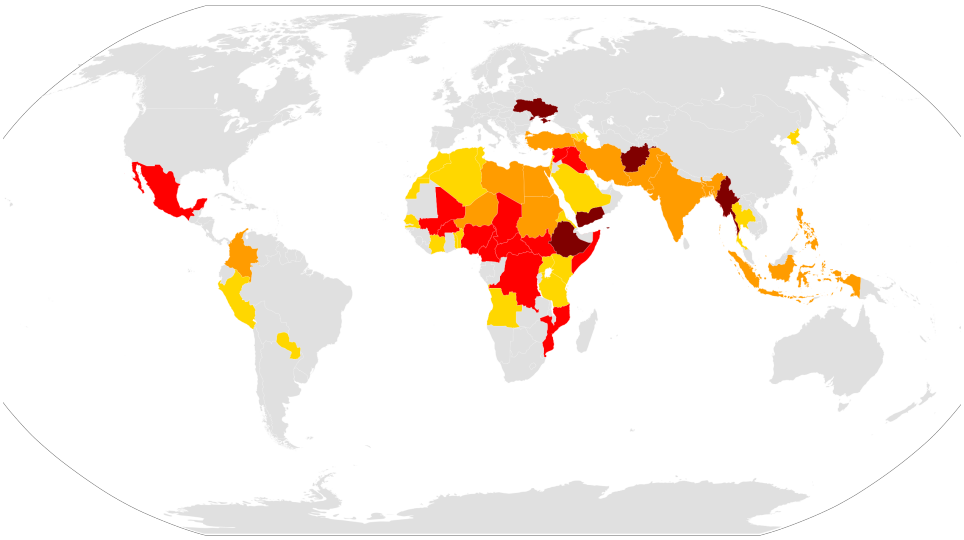


Figura 2.1: Mappa dei conflitti armati attualmente in corso. Immagine tratta da Wikipedia.

Network) la cui estensione può variare a seconda delle caratteristiche del territorio, delle tecnologie utilizzate per crearla (e.g. router in cascata, access point in modalità bridge, etc.) e dal numero degli abitanti della zona. Grazie all'utilizzo di una blockchain locale, gli individui che abitano in luoghi isolati possono utilizzare sistemi di pagamento digitali e servizi su blockchain anche in contesti in cui sarebbe impossibile impiegare una blockchain come Ethereum a causa dell'assenza di Internet Service Provider che coprano la zona o qualora i contratti da essi offerti risultassero economicamente sconvenienti.

2.1.2 Scenario industriale

Le blockchain locali possono essere impiegate anche a livello industriale; infatti, oggi più che mai, la confidenzialità delle informazioni aziendali è un tema di fondamentale importanza per le aziende. L'utilizzo di blockchain in ambito aziendale, come già accennato in precedenza, si trova ancora in una fase piuttosto embrionale, forse anche a causa di una ritrosia delle aziende a utilizzare tecnologie in cui i dati sono tendenzialmente pubblici; tuttavia nuove opportunità di utilizzo possono nascere dalla disponibilità di blockchain raggiungibili solo dall'interno del network aziendale.

La presenza di chain non direttamente accessibili al pubblico potrebbe convincere le aziende ad integrare attivamente questa tecnologia in vari ambiti aziendali. Per quanto le aziende possano sfruttare chain pubbliche, proteggendo i dati a livello applicativo, o blockchain private, in cui solo alcuni nodi hanno privilegi di scrittura, disporre di una soluzione "ibrida", in cui qualsiasi dipendente può istanziare un proprio nodo, può aiutare l'azienda ad ottenere una maggiore decentralizzazione della propria blockchain, a diffondere una maggior consapevolezza sul funzionamento delle blockchain tra i propri dipendenti e a disporre di un ambiente in cui operare sperimentazioni, utilizzabile da chiunque abbia accesso all'intranet aziendale, con la garanzia di mantenere le informazioni che transitano sulla chain invisibili agli occhi delle aziende concorrenti.

Alcune opportunità innovative che possono nascere dall'aggiunta delle blockchain ai servizi offerti dall'intranet aziendale possono essere:

- uso della pseudoanonimità garantita dalla crittografia per introdurre tecniche innovative di governance aziendale (e.g. effettuare proposte e votarle senza che vi siano bias derivanti dall'identità dell'individuo che effettua la proposta e dal ruolo che esso ricopre in azienda);
- supportare fenomeni di whistleblowing consentendo di segnalare pseudoanonimamente comportamenti contrari all'etica aziendale;
- creare un sistema di crediti aziendali utilizzabili, ad esempio, per il pagamento di servizi interni (e.g. corsi di formazione, mensa) o per l'interazione con smart contract realizzati da altri dipendenti;
- monitoraggio mediante blockchain dei processi produttivi aziendali;
- chiusura e documentazione di contratti via blockchain.

2.2 Personas

Descritti alcuni possibili utilizzi delle blockchain locali, verranno ora descritte alcune personas[11] per dettagliare più precisamente le necessità e le aspettative che i potenziali utenti avrebbero nei confronti del sistema nel mondo reale.

2.2.1 Prima Persona

Attore	Alessandro: Gestore di un rifugio
Scenario	Alessandro gestisce un rifugio con ristorante in una zona montuosa del Nord Italia. Date le caratteristiche del territorio, non è disponibile alcuna rete mobile presso il rifugio. Alessandro potrebbe concludere un contratto per utilizzare internet via satellite, tuttavia le compagnie esistenti offrono piani dati a prezzi poco competitivi; inoltre l'utilizzo pianificato da Alessandro è legato puramente a pagamenti digitali, pertanto la spesa per l'installazione della parabola non sarebbe giustificata e comunque rientrare dell'investimento richiederebbe tempo.
Analisi	I clienti di Alessandro possono effettuare pagamenti digitali connettendosi al Wi-Fi del suo locale per interagire con la blockchain locale. Una volta connessi, i clienti possono inviare un pagamento all'address di Alessandro utilizzando il loro wallet.

2.2.2 Seconda Persona

Attore	Benedetta: Gestore di un museo

[11] Per ulteriori informazioni sulle personas e sui loro utilizzi si rimanda a: `https://www.usability.gov/how-to-and-tools/methods/personas.html`.

Scenario	Benedetta gestisce una piccolo museo sulle dolomiti, dispone di un POS, ma la connettività mobile è piuttosto inaffidabile. Oltre ad esserci problemi di ricezione a causa della posizione del museo e delle mura particolarmente spesse, soventemente le intemperie rendono impossibile stabilire una connessione col gestore del POS. Benedetta vorrebbe inoltre poter rendere più attraente il museo anche a giovani visitatori; la sua idea è quella di fare in modo che chi visita il muso possa riscattare gratuitamente un NFT che attesti di essersi recato fisicamente alle mostre o ai numerosi eventi culturali che ciclicamente vengono proposti.
Analisi	I visitatori del museo di Benedetta possono connettersi al Wi-Fi del museo e pagare il biglietto utilizzando la blockchain locale. Il riscatto dell'NFT relativo alla mostra può avvenire chiamando la funzione mint all'interno dello smart contract preposto alla distribuzione dei token; anche in questo caso lo smart contract è raggiungibile solamente dopo essersi connessi alla rete locale del museo.

2.2.3 Terza Persona

Attore	Asfodel: restauratrice, Lamin: Negoziante

Scenario	Asfodel è una restauratrice impegnata in un progetto in un paese del Medio Oriente. Per quanto la zona in cui lavora Asfodel sia sicura, alcune proteste nella capitale hanno causato disagi alle linee telefoniche, inoltre l'interazione con alcuni domini è stata bloccata dall'Internet Service Provider del paese. Tra i domini interessati dal blocco vi sono alcuni noti circuiti bancari, emittenti delle carte di credito utilizzate dai componenti del team di restauro, di conseguenza i pagamenti in valuta estera spesso incontrano problemi nell'essere processati. Lamin è uno dei negozianti del paese in cui si svolgono i restauri, il suo Bar è vicino alla zona in cui si svolgono i restauri e non vuole perdere la possibilità di offrire i suoi servizi a coloro che lavorano al progetto di restauro.
Analisi	Similmente ai casi di Alessandro e Benedetta, gli avventori del Bar di Lamin possono connettersi alla sua rete locale e usare la blockchain locale per pagare. Altri negozianti possono sfruttare la stessa blockchain locale invitando i clienti a pagare presso il Bar di Lamin o istanziando anche loro un nodo e unendosi alla blockchain.

2.2.4 Quarta Persona

Attore	Elia: Titolare di un'azienda informatica

Scenario	Elia è direttore di una grande azienda che offre prodotti software ad aziende che operano in svariati campi (automotive, moda, etc.). Elia, dopo alcuni problemi di pirateria nei confronti del suo software, vorrebbe un sistema contro la contraffazione delle licenze. L'utilizzo di Token Non Fungibili potrebbe consentire di associare ad ogni licenza venduta un NFT che viene coniato sulla blockchain aziendale; inoltre sperimentare in questo campo permetterebbe a Elia di avvantaggiarsi rispetto alla concorrenza. Elia vorrebbe poi estendere il sistema per utilizzarlo anche con i suoi fornitori, in questo modo potrebbe garantire una filiera completamente tracciabile; tuttavia non vuole che le informazioni siano visibili da chiunque, la visibilità deve essere garantita solo a persone che operano all'interno del network aziendale.
Analisi	Elia può verificare che il codice di licenza fornito dai clienti sia effettivamente presente tra gli NFT rilasciati nella sua blockchain. I fornitori possono partecipare alla blockchain di Elia aggiungendo dei nodi da loro gestiti che si connettono all'infrastruttura aziendale via VPN.

2.3 Modello di sistema

Dall'analisi condotta mediante l'utilizzo delle personas emerge come le blockchain locali possano aiutare a raggiungere l'obiettivo dei protagonisti degli scenari, tuttavia è anche possibile notare come i sistemi di cui si è discusso rimangano al momento chiusi e isolati.

L'utilià delle blockchain locali è massimizzata qualora questi sistemi si "aprano" all'esterno, ad esempio: i clienti di Alessandro, Benedetta e Lamin difficilmente vorranno effettuare acquisti utilizzando delle valute che abbiano un valore solo all'interno delle blockchain dei locali che visitano, i destinatari degli NFT commemorativi delle mostre di Benedetta potrebbero voler spostare i propri token sulle blockchain che usano nor-

malmente e, infine, i fornitori di Elia potrebbero voler sviluppare soluzioni autonome per poi interfacciarle con quelle di terze parti per evitare lock-in. È quindi un requisito fondamentale che le blockchain locali non siano autoreferenziali, ma possano comunicare tra loro ed anche con chain pre-esistenti (e.g. Ethereum, Bitcoin, etc.), permettendo lo scambio di token fungibili e non fungibili, che diventano quindi agnostici rispetto alla chain di provenienza.

L'obiettivo appena presentato è raggiungibile facendo utilizzo di tecnologie che consentano l'interoperabilità tra blockchain; a partire da una qualsiasi blockchain, sia essa locale o meno, è quindi possibile sfruttare una specifica implementazione delle tecniche presentate nella sezione 1.4 per trasferire i propri token nei luoghi che si intende visitare, nei quali non è raggiungibile una chain non locale, e utilizzarli per acquistare beni e servizi fino a che non si decide di lasciare la zona servita solo dalla blockchain locale.

Nello specifico scenario in analisi, dovendo scegliere tra varie tecniche per l'interoperabilità, impieghiamo quella che garantisce la maggior flessibilità nella comunicazione tra chain, ovvero i Notary Schemes. L'utilizzo di questa tecnica, a differenza degli HTLC, consente infatti di trasferire sia dati sia asset; inoltre, a differenza dei Relays, permette di non dover scambiare tra le blockchain lo storico degli header dei blocchi, operazione che potrebbe essere particolarmente lunga in caso tra due chain locali non vi fosse stata alcuna interazione per un periodo di tempo prolungato.

Il flusso di un trasferimento di token, considerando un generico sistema basato su Notary Scheme, è quindi il seguente:

1. un utente effettua il deposito dei token che vuole utilizzare presso un bridge[12] della propria blockchain configurato per inviare depositi verso la chain locale che si intende raggiungere;

2. un nodo della blockchain su cui è stato effettuato il deposito si disconnette dal network della chain di origine dell'utente e raggiunge il network della blockchain di destinazione, connettendovisi;

[12]Con il termine bridge indichiamo uno smart contract che funge da "punto di accesso" per inviare i propri token ad un altra blockchain.

3. l'evento di deposito viene trasmesso al bridge della chain locale mediante il relayer[13];

4. vengono accreditati i token sul wallet dell'utente presso la chain di destinazione;

5. l'utente può utilizzare i propri token.

Nella figura 2.2 è visibile il processo appena descritto.

2.3.1 Deploy dei bridge

Perchè vi possa essere comunicazione tra le chain locali è necessario che sulla blockchain di origine e destinazione siano correttamente istanziati i bridge per collegare i due sistemi. Assumendo quindi che il deploy e la configurazione del bridge vengano svolti direttamente dagli individui che operano all'interno delle blockchain locali interessate a stabilire una connessione, tali individui ricopriranno il ruolo di gestori del bridge.

Un aspetto da tenere in considerazione nel deploy dei bridge è che la presenza di molte blockchain locali non richiede necessariamente che queste siano tutte interconnesse tra loro; infatti è probabile che la maggior parte delle chain locali sia connessa direttamente a catene non locali, come Ethereum, in quanto direttamente raggiungibili da un centro abitato in cui è disponibile una connessione affidabile.

Il caso di blockchain locali direttamente connesse tra loro è invece più probabile in zone in cui gli spazi privi di connessione o con connessione scadente e instabile coprano aree più ampie. In altre parole, il quantitativo di bridge in uscita e in ingresso da una chain locale dipende fortemente da quanto essa sia isolata rispetto al primo punto con connettività affidabile, da cui quindi è contattabile una blockchain non locale; è quindi verosimile che il network dei bridge presenti alcune chain "hub" da cui è possibile raggiungere la maggior parte delle chain locali.

[13]Il relayer è tipicamente un software che effettua il parsing degli eventi emessi dalla blockchain di origine e li inoltra, mediante una serie di transazioni, allo smart contract bridge della chain di destinazione. Generalmente il relayer è sotto il controllo di un individuo che ne è responsabile del corretto funzionamento, pertanto il termine può anche essere utilizzato per identificare il ruolo di certificatore dei depositi e i suoi privilegi.

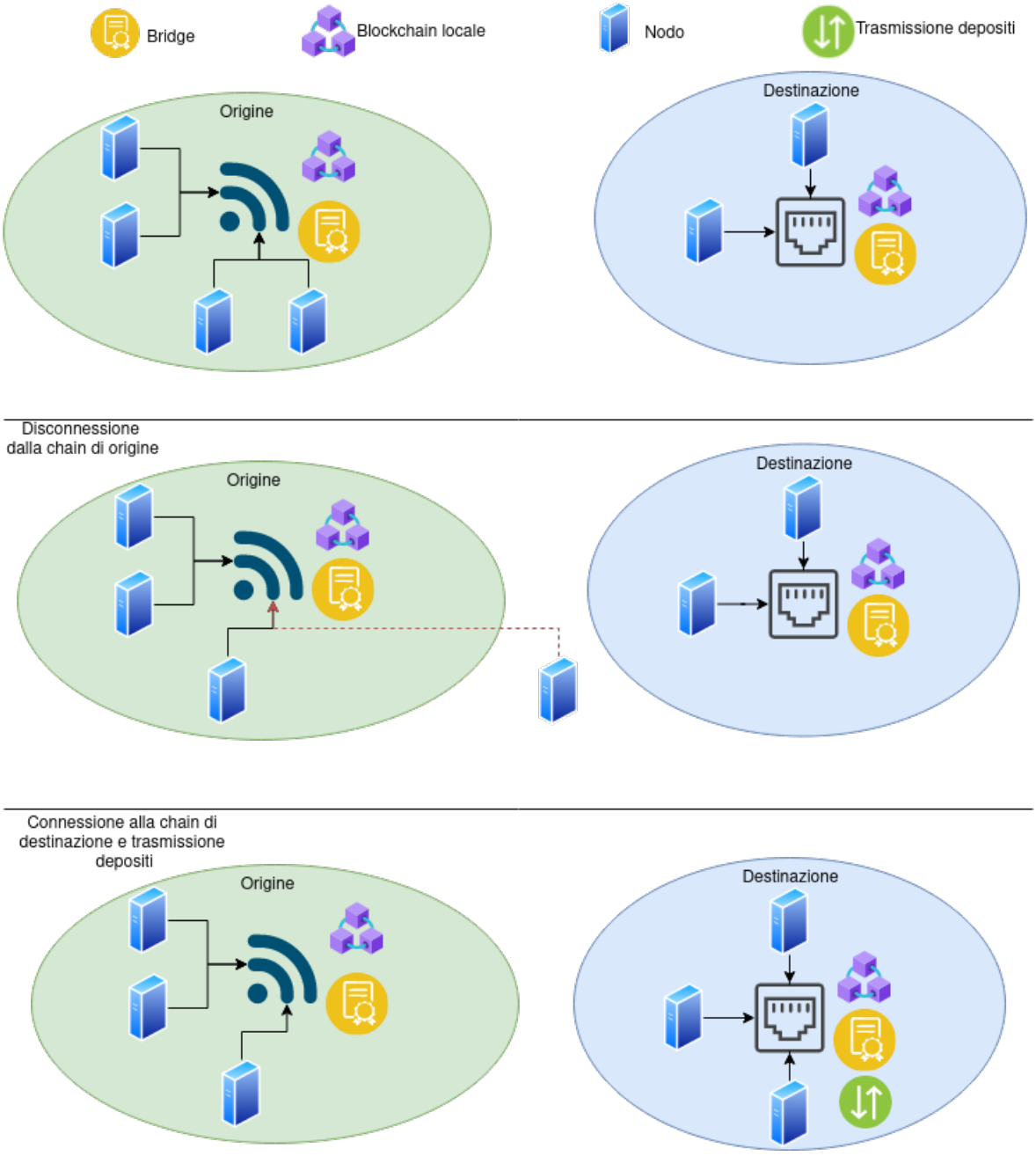


Figura 2.2: Trasferimento di token tra chain locali.

2.3.2 Raggiungimento blockchain di destinazione

Uno degli aspetti fondamentali del processo di trasmissione dei propri token riguarda la fase di disconnessione di un nodo dalla propria blockchain di origine e il raggiungimento della blockchain di destinazione.

Nel caso di uno scenario di utilizzo rurale, il nodo della blockchain di origine del deposito deve fisicamente raggiungere la LAN della blockchain di destinazione. Ciò può avvenire in due modi: l'utente, con un computer, si reca personalmente presso la destinazione in cui è accessibile la rete locale wireless o wired associata alla chain locale, avvia il suo nodo e procede alla trasmissione dei propri eventi; la seconda opzione consiste invece nel delegare questa operazione a terzi. Nel caso appena descritto è possibile ipotizzare modalità più strutturate che garantiscano periodicamente la trasmissione dei depositi su una certa tratta, ad esempio, utilizzando mezzi di trasporto come data mule.

L'utilizzo di veicoli (pullman, fuoristrada che servono rifugi di montagna, cabinovie, etc.) consente di trasportare le informazioni relative ai depositi degli utenti senza che questa operazione debba essere effettuata da loro; tale soluzione consente di sfruttare veicoli, che dovrebbero comunque coprire determinate tratte per altre motivazioni (e.g. trasporto di persone e cose), per svolgere il deposito dei fondi degli utenti anticipatamente rispetto al loro arrivo presso le chain locali di destinazione. Un esempio pratico di quanto descritto è il seguente: un escursionista che risiede in una zona turistica ben servita dalla connessione, dopo aver pianificato le proprie escursioni, effettua una serie di depositi per ogni rifugio in cui intende effettuare spese. Nei giorni successivi chi si recherà (individui, veicoli, etc.) presso le chain locali destinatarie del deposito, trasmetterà il deposito dell'utente, che, quando arriverà a sua volta a destinazione, disporrà immediatamente dei token inviati nei giorni precedenti. Un sistema capillare per la trasmissione dei depositi richiede un'approfondita fase di sperimentazione e affinamento del servizio affinchè questo funzioni adeguatamente a pieno regime, inoltre è anche richiesta la collaborazione da parte delle aziende di trasporto, individui terzi, etc., per la sua erogazione; la disponibilità del servizio potrebbe comunque avvenire inizialmente per un piccolo numero di tratte, da ampliare poi gradualmente. Inoltre esistono già studi sull'utilizzo di data mule per l'interazione con le blockchain in scenari di scarsa connettività che possono fungere da

base di partenza per la progettazione e l'implementazione del sistema[14].

Per quanto riguarda il caso industriale, la fase di raggiungimento della chain di destinazione è decisamente semplificata. Supponendo che due aziende dispongano di due blockchain locali nei rispettivi network, è sufficente fare utilizzo di tecnologie come una VPN per permettere che un nodo della blockchain di origine dei depositi sia visibile dal network della seconda azienda; a questo punto è possibile procedere alla trasmissione dei depositi allo stesso modo di quanto descritto in precedenza.

2.3.3 Trasmissione dei depositi

La trasmissione dei depositi alla blockchain di destinazione avviene una volta che il nodo contenente lo storico delle transazioni della chain di origine raggiunge una zona coperta dalla rete locale. Considerando che l'operazione di trasmissione dei depositi interessa gli utenti, verrà a loro delegata la gestione e configurazione del relayer.

Come accennato in precedenza, nei Notary Schemes è presente un elemento di fiducia relativo all'onestà dei relayer, ossia gli utenti che certificano gli avvenuti depositi. Gli scenari di utilizzo descritti nelle sezioni precedenti prendono in considerazione aree rurali in cui il flusso di persone è tutto sommato limitato o incostante; questo non permette di impostare bridge che richiedano un numero elevato di conferme da parte di relayer differenti, in quanto sarebbe necessario attendere che altri utenti provenienti dalla stessa chain di origine dell'utente giungessero con un loro nodo e un relayer per trasmettere a loro volta depositi e votare quelli avvenuti in precedenza. La situazione appena descritta potrebbe essere accettabile per rifugi di montagna, in cui il numero di visitatori in certi periodi dell'anno è elevato, tuttavia lo stesso discorso non è detto sia valido anche per zone particolarmente remote; inoltre, variabili come il mal tempo possono influire negativamente sull'afflusso di individui presso una zona.

Condizioni più o meno restrittive a livello di conferma di depositi possono essere impostate dai gestori dei bridge; tuttavia questo comporta un maggiore tempo di attesa prima che gli utenti possano utilizzare i loro token ed aggiunge un elemento di incertezza sulla possibilità che il deposito venga effettivamente finalizzato. Decidendo di optare per la finalizzazione di un deposito a seguito di un singolo voto, si rende necessario svolgere

[14]S. Ferretti et al., *Incentivized Data Mules Based on State-Channels*, IEEE, 2022.

un'attenta valutazione dei risvolti che ciò può avere sul sistema e se questo permetta di effettuare degli attacchi ai danni dell'architettura appena teorizzata.

Ulteriori problematiche relative ai depositi potrebbero derivare dalla scarsa decentralizzazione delle blockchain locali. Infatti, considerando che esse servono zone con espansione territoriale relativamente contenuta, gli individui che vi abitano, e che pertanto gestiranno un nodo, saranno verosimilmente un numero limitato; è quindi possibile che le blockchain locali presentino un grado di centralizzazione piuttosto elevato, derivante da uno scarso numero di nodi nella rete.

Le due condizioni appena descritte, unite alla totale autonomia con cui un bridge può essere istanziato tra due chain, può avere un impatto difficilmente prevedibile sulla sicurezza dell'utilizzo dei bridge tra blockchain, visto e considerato che attacchi a questo tipo di infrastrutture avvengono con successo anche nei confronti di bridge che operano in contesti meno particolari di quello appena teorizzato.

Alla luce di quanto detto, si rende quindi necessario svolgere un'attenta analisi dei possibili vettori d'attacco che potrebbero essere sfruttati a partire dalle specifiche condizioni in cui gli utenti del sistema si trovano ad operare.

2.4 Deploy sistema

Dettagliato lo scenario d'uso e il modello di sistema, procediamo ora al deploy di un prototipo del sistema, che verrà utilizzato per testare l'effettiva impiegabilità della soluzione proposta e analizzarne le possibili vulnerabilità. Le tecnologie utilizzate sono due: il Polygon-Edge SDK, impiegato per la creazione delle blockchain locali, e ChainBridge, per il trasferimento dei token tra le chain.

2.4.1 Hardware e configurazione delle blockchain

La creazione di uno scenario realistico per lo svolgimento dei test sugli attacchi consiste nell'istanziazione di tre blockchain, ognuna delle quali formata da quattro nodi validatori. Il numero di validatori, tenuto volutamente limitato, deriva dall'ipotesi di scarsa distribuzione delle catene a causa della bassa popolosità dei centri abitati teorizzati nella sezione 2.3.

Il processo di inizializzazione[15] delle blockchain consiste fondamentalmente in quattro passaggi:

1. installazione del Polygon-Edge SDK sui nodi facenti parte della chain;
2. generazione delle chiavi per ogni validatore;
3. generazione della stringa di connessione in formato multiaddress comprendente indirizzo ip, porta e `node_id` del nodo validatore[16];
4. generazione del file genesis contenente le informazioni necessarie all'inizializzazione della blockchain. I dati vengono firmati mediante chiave pubblica dei validatori; in questo modo ogni nodo validatore può verificare che lo stato iniziale del primo blocco della catena contenga informazioni coerenti con quelle decise nel momento della creazione della blockchain.

Considerata la ripetitività delle operazioni da effettuare per ogni nodo (installazione Polygon-Edge SDK, generazione chiavi validatori e stringa multiaddress) e per ogni catena (generazione file genesis), la procedura di inizializzazione delle blockchain è stata automatizzata mediante uno script in Python che permette di configurare le blockchain senza che l'utente debba svolgere manualmente ognuna delle operazioni precedentemente citate[17].

L'hardware utilizzato per ospitare i 12 container dei nodi validatori consiste in un HP MicroServer n34l con CPU AMD Athlon II Neo@1.3GHz e 16GB di RAM. L'assegnazione delle risorse ai container, che consistono in 2 core e 512MB di ram ad ogni container, viene gestita utilizzando come hypervisor Proxmox. Con questa allocazione di risorse

[15]Per una completa disamina di tale processo si rimanda alla documentazione ufficiale del Polygon-Edge SDK.

[16]Il Polygon-Edge SDK utilizza la libreria libp2p per la comunicazione tra nodi, pertanto le informazioni di connessione vengono distribuite utilizzando il formato multiaddress. Il formato in questione consente di specificare diverse informazioni di indirizzamento in un'unica stringa "autodescrivente" semplicemente leggibile dall'uomo e processabile da calcolatori. Il formato è definito nel modo seguente: `(/<protocol_name>/<value>)+`. Un esempio pratico di utilizzo è quindi: `/ip4/127.0.0.1/tcp/80`. Per un'accurata disamina del formato si rimanda a `https://docs.libp2p.io/concepts/addressing/` e `https://github.com/multiformats/website/blob/master/content/multiaddr.md`.

[17]Lo script è pubblicamente disponibile presso `https://github.com/NorwegianGoat/edge_utils`.

è stato lanciato un benchmark mediante `loadbot`[18] da cui è emerso come, con questa configurazione, ogni blockchain supporti stabilmente fino a 200 transazioni per secondo; il risultato deriva chiaramente dal protocollo di consenso utilizzato, ossia l'Istanbul Bizantine Fault Tolerance (IBFT)[19].

Considerando le scarse risorse hardware assegnate ad ogni container, i risultati ottenuti sono assolutamente positivi e ciò rende adatto il Polygon-Edge SDK all'utilizzo in ambito rurale; infatti l'uso di hardware particolarmente energivoro e di classe enterprise non è probabilmente una opzione praticabile in zone del mondo in via di sviluppo.

2.4.2 Configurazione e funzionamento di ChainBridge

ChainBridge[20] è un progetto open source[21] che consente di interconnettere blockchain EVM compatibili.

Seguendo la tassonomia stilata nella sezione 1.4, ChainBridge rientra all'interno dei Notary Schemes; questa soluzione supporta entrambe le modalità di interoperabilità dati e asset e non richiede modifiche alle blockchain esistenti, in quanto il sistema si basa su alcuni smart contract scritti in Solidity e su un software realizzato in Go che, eseguito off-chain, monitora gli eventi lanciati dagli smart contract di ChainBridge.

Il funzionamento di ChainBridge è piuttosto articolato e richiede la descrizione di alcuni smart contract e dei ruoli e privilegi assegnati agli utenti. Pertanto la descrizione delle varie componenti di ChainBridge verrà svolta contemporaneamente alla configurazione dei bridge tra le blockchain appena istanziate; in questo modo sarà possibile illu-

[18]L'utility di benchmark è integrata nel Polygon-Edge SDK, per ulteriori informazioni si rimanda alla documentazione del Polygon-Edge SDK.

[19]Una completa descrizione del funzionamento di questo protocollo è reperibile presso: `https://github.com/ethereum/EIPs/issues/650`. La scelta di utilizzare IBFT piuttosto che altri metodi di consenso deriva dal fatto che, al momento dell'istanziazione delle blockchain, l'IBFT era l'unico protocollo supportato dal Polygon-Edge SDK; il team di sviluppo di Polygon sta tuttavia procedendo attivamente allo sviluppo della piattaforma allargando il supporto ad altri protocolli di consenso, come, ad esempio, il Proof Of Stake.

[20]Per ulteriori risorse si rimanda al sito web della piattaforma: `https://chainbridge.chainsafe.io/`.

[21]Il repository contenente gli smart contract di ChainBridge è pubblicamente disponibile presso: `https://github.com/ChainSafe/chainbridge-solidity/tree/master/contracts`.

strare l'interazione tra i vari contratti e alcune delle funzionalità fondamentali associate ad ogni attore del sistema. Anche in questo caso la configurazione è stata automatizzata mediante una serie di script in Python.

Il bridge admin, ossia l'utente che disporrà dei privilegi di amministrazione del bridge, effettua il deploy di due contratti sulla chain di origine dei token e sulla chain di destinazione: i contratti in questione sono il `Bridge` e l'`ERC20Handler`[22].

Il `Bridge` offre un punto di ingresso a tutte le funzionalità offerte da ChainBridge, di cui le fondamentali sono: depositare token da trasferire presso un'altra blockchain, configurare Fee per gli utilizzatori del sistema, associare contratti al Bridge (i.e. mappare il bridge), autorizzare o rimuovere relayer[23]. e modificare il numero dei voti richiesti per ultimare un deposito. In fase di deploy del `Bridge` vengono passati in input al costruttore dello smart contract gli indirizzi dei relayer autorizzati a trasmettere eventi di deposito e a votarli nonchè il threshold di voti necessari a considerare un deposito ultimato. Considerando che la riuscita di un deposito dipende unicamente dal numero di voti positivi inviati dai relayer, è di fondamentale importanza che la funzione per il voto del deposito sia richiamabile solamente da un ristretto set di indirizzi fidati; da ciò deriva la necessità dell'elenco di relayer autorizzati impostati dall'admin. Nel caso del modello di sistema da noi ipotizzato in precedenza, il bridge admin dovrà autorizzare l'indirizzo di ogni utente che voglia effettuare un deposito.

L'`Erc20Handler` è il contratto che funge da ponte tra il `Bridge` e lo smart contract che gestisce i token ERC-20; la necessità di avere un contratto intermedio che svolga azioni per conto del `Bridge` deriva dal fatto che, a seconda del tipo di token per cui un bridge[24] viene configurato, possono essere implementate procedure differenti.

Effettuato il deploy degli smart contract, l'admin richiama la funzione `adminSetResource` sul contratto `Bridge` fornendo: l'indirizzo dell'`ERC20Handler` da utilizzare, l'indirizzo del

[22] I test inerenti alle possibili vulnerabilità del sistema verranno effettuati utilizzando un token ERC-20; tuttavia ChainBridge permette di trasferire anche Token Non Fungibili.

[23] In questa sezione verranno utilizzati più volte i termini relayer e relay, anche se, come già accennato in precedenza, non indicano una tecnologia per consentire l'interoperabilità tra blockchain, ma un ruolo nel sistema e una componente software per il parsing degli eventi.

[24] In questo caso con il termine bridge si intende il "ponte" che si crea tra una blockchain e un'altra e non lo smart contract utilizzato per interagirvi. Per distinguere i due concetti ci si riferirà allo smart contract utilizzando il font monospaziato e la prima lettera maiuscola.

token ERC-20 per cui si vuole creare il bridge e un `resourceID` da assegnare al bridge relativo a questo token. La presenza del `resourceID` potrebbe sembrare superflua, considerando che per distinguere tra più token registrati presso lo stesso `Bridge` si potrebbe utilizzare solo l'address degli smart contract; tuttavia l'utilizzo di un ID si rende necessario in quanto uno stesso token ERC-20 potrebbe essere trasferibile verso più chain. Utilizzando lo stesso `Bridge` e facendo uso di un ID è quindi possibile specificare quale bridge si vuole utilizzare e, quindi, verso quale network si vuole inviare il token.

Un ulteriore parametro da impostare sul `Bridge` riguarda se i token di cui gli utenti effettuano il deposito presso la blockchain di origine vadano distrutti o semplicemente bloccati, ricadendo sotto il bilancio dell'`ERC20Handler`. L'impostazione di default di ChainBridge prevede il blocco di token; qualora il Bridge Admin volesse modificare questo comportamento è possibile chiamare la funzione `adminSetBurnable` fornendo l'indirizzo dell'handler di cui modificare l'impostazione e l'indirizzo del token per cui si vuole impostare la nuova modalità. Optare per la distruzione dei token ad ogni deposito rende però necessario concedere i privilegi di mint dei token all'`ERC20Handler` sul contratto ERC-20 presso la chain di destinazione, in quanto, in caso contrario, non sarebbe possibile creare *ex novo* gli asset depositati dall'utente; questa operazione, a seconda dell'implementazione del contratto ERC-20, può richiedere la collaborazione del creatore del token, che potrebbe essere l'unico ad avere i privilegi necessari ad aggiungere indirizzi autorizzati ad effettuare il mint dei token.

Configurato adeguatamente il bridge sulla chain di origine e sulla blockchain di destinazione, è necessario installare il software relativo al relay e impostarlo mediante un file JSON. Il file di configurazione contiene: gli indirizzi dei `Bridge` di cui il relay deve svolgere il polling degli eventi, il `resourceID` della risosrsa da monitorare e gli endpoint dei nodi delle blockchain tramite cui ottenere gli eventi.

Terminata la configurazione del relay, vediamo ora il flusso di esecuzione di un deposito[25].

L'attore iniziatore di un deposito è l'utente del bridge, che avvia la procedura autorizzando, mediante il contratto ERC-20, l'`ERC20Handler` a gestire un determinato quan-

[25]Alcuni dei parametri necessari allo svolgimento di un deposito verranno omessi per alleggerire e semplificare la spiegazione della procedura, tuttavia ciò non ne cambia minimamente lo svolgimento.

titativo dei suoi token; l'autorizzazione viene effettuata chiamando la funzione `approve` e fornendo l'address dell'handler insieme al quantitativo di token che lo si autorizza a gestire. A seguito dell'autorizzazione, l'utente chiama la funzione `deposit` sul `Bridge`, fornendo il `resourceID` associato al token che si vuole depositare, il numero di token e il beneficiario del deposito. Il `Bridge` recupera quindi l'handler associato al token dell'utente e passa i parametri forniti da quest'ultimo alla funzione `deposit` dell'`ERC20Handler`, che, in caso di bridge in modalità lock, chiamerà la funzione transfer sul contratto relativo al token, intestando all'handler gli asset che l'utente ha depositato, mentre, in caso di bridge in modalità burn, chiamerà la funzione `burnFrom` per distruggere i token[26].

Se il procedimento appena descritto non ha prodotto alcun genere di errore, il `Bridge` lancia l'evento `Deposit`, contenente tutti i dati relativi al deposito dell'utente. I relayer che, come detto in precedenza, effettuano il polling dei blocchi e ne processano gli eventi alla ricerca di depositi, recuperato il blocco contenente l'evento di deposito, procedono a inviare il proprio voto riguardo al deposito sulla blockchain di destinazione. L'invio del voto avviene mediante il metodo `voteProposal`, che, tra le prime operazioni svolte, verifica che la chiamata provenga da un indirizzo autorizzato; qualora i controlli sull'identità del votante dessero esito positivo, il voto verrà salvato, mentre, in caso contrario, l'esecuzione del metodo terminerà con un errore. Al raggiungimento del quantitativo minimo richiesto di voti positivi, uno dei relayer potrà chiamare la funzione `executeProposal`; tale funzione consente di procedere con il mint dei nuovi token, qualora il `Bridge` sia in modalità burn/mint o, in caso contrario, con lo sblocco da quelli sotto il possesso dell'`ERC20Handler`.

La figura 2.3 rappresenta il procedimento appena descritto, mentre l'immagine 2.4 raffigura l'andamento del bilancio sulla blockchain di origine e di destinazione prima e dopo la procedura di deposito.

[26]Ricordiamo che l'`ERC20 Handler` è in grado di gestire i token dell'utente solo in quanto precedentemente autorizzato mediante la funzione `approve`.

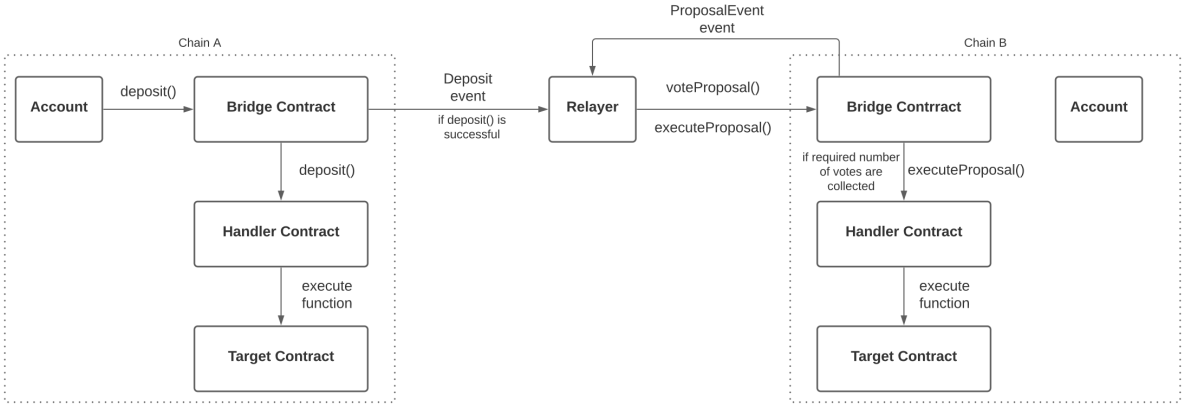
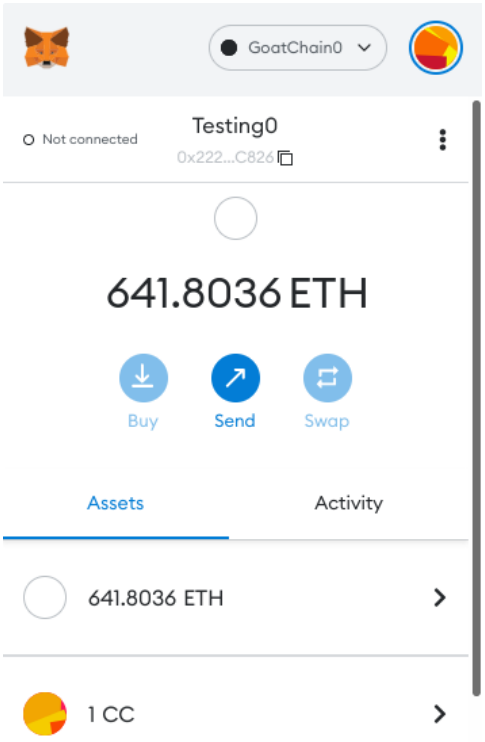
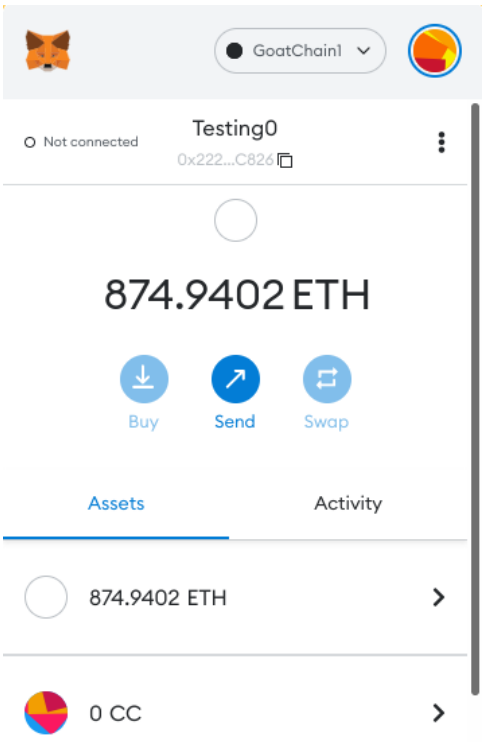


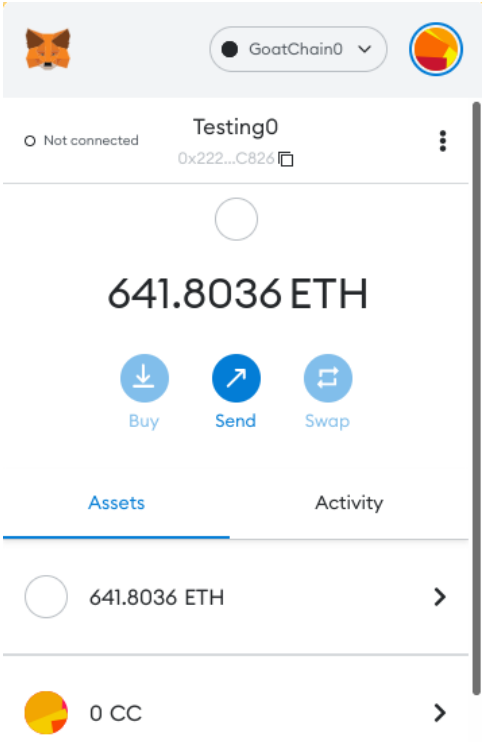
Figura 2.3: Flow esecuzione di un deposito. Immagine tratta dalla documentazione di Polygon Edge.



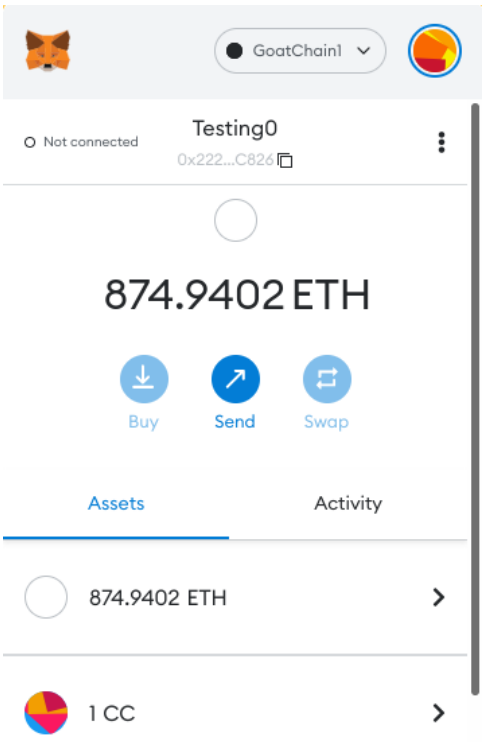
(a) Bilancio sulla blockchain0 prima della procedura di deposito.



(b) Bilancio sulla blockchain1 prima della procedura di deposito.



(c) Bilancio sulla blockchain0 al termine della procedura di deposito.



(d) Bilancio sulla blockchain1 al termine della procedura di deposito.

Figura 2.4: Andamento del bilancio di un utente prima e dopo la procedura di deposito di token tra blockchain.

Capitolo 3

Attacchi architetturali e soluzioni

In questo capitolo affrontiamo la descrizione di sei attacchi, che, a seguito di un'analisi delle vulnerabilità nello scenario ipotizzato nella sezione 2.3, sono stati svolti con successo nei confronti del sistema. A seguito della descrizione di ogni attacco verranno proposti alcuni requisiti per la risoluzione o, quantomeno, mitigazione delle vulnerabilità che lo rendono possibile.

Gli attacchi che saranno discussi nelle sezioni a seguire rientrano in tre macro categorie: fund stealing, double spending e bridge poisoning. La prima categoria prevede che i token trasferiti dagli utenti vengano loro sottratti indebitamente; la seconda, che gli stessi fondi vengano trasferiti da una blockchain a un'altra più volte, creando di fatto una doppia spesa dei token; infine, l'ultima categoria prevede un'interazione col bridge volta a trasmettere eventi di deposito da fonti non attendibili.

Passiamo ora alla descrizione del meccanismo sfruttato per simulare lo scenario in cui si svolgono le operazioni di deposito. Tutti gli attacchi testati prendono in considerazione un utente che, con un nodo e il relayer, si reca dalla sua blockchain di origine ad una chain di destinazione, nella quale egli ha qualche interesse (e.g. turismo, acquisto di beni o servizi locali, etc.); in sostanza lo scenario simulato è quello di una zona rurale, ma quanto svolto risulterebbe valido anche qualora si volesse simulare lo scenario industriale ipotizzato nella sezione 2.1.2.

Lo spostamento tra blockchain e l'irraggiungibilità dei nodi della chain di destinazione (prima dello spostamento) e di quelli di origine (a seguito del raggiungimento della

destinazione) vengono simulate mediante l'utilizzo di regole del firewall impostate sul container rappresentante il computer dell'utente che si sposta; dapprima verranno bloccate le connessioni verso i nodi della chain di destinazione, in seguito al raggiungimento della meta verranno poi bloccate le connessioni verso la chain di origine e sbloccati i nuovi nodi locali.

Le modifiche al firewall vengono gestite da uno script in Python, che esegue i test relativi agli attacchi e che effettua alcune chiamate a una classe wrapper di UFW[1] realizzata *ad hoc*[2].

Una volta raggiunta la chain di destinazione viene avviato il relayer e gli eventi generati sulla blockchain di origine vengono trasmessi al bridge della chain locale presso cui l'utente si è trasferito. Per velocizzare la fase di test degli attacchi, in fase di deploy dei bridge sono stati forniti gli indirizzi dei wallet degli utenti utilizzati nelle varie prove; in questo modo viene evitata la fase di aggiunta e rimozione dell'utente tra i relayer autorizzati ad inviare depositi su una determinata blockchain locale, operazioni che altrimenti andrebbero effettuate ad ogni test.

Al termine dei depositi viene poi controllato il bilancio dell'utente per verificare l'esito dell'attacco. Infine si effettua il ripristino delle regole del firewall del container allo stato precedente (i.e. l'utente viene "reinserito nella propria chain di origine") in modo da testare ogni attacco a partire dalle stesse condizioni.

3.1 CrossCoin Theft Attack

Il CrossCoin Theft Attack rientra negli attacchi di categoria fund stealing, in quanto i token di cui l'utente effettua il deposito vengono accreditati a terzi.

L'attacco si svolge nel seguente modo:

[1]Per ulteriori informazioni su Unconmplicated Firewall si rimanda a: `https://wiki.ubuntu.com/UncomplicatedFirewall`.

[2]Il repository contenente le risorse relative agli attacchi è diverso rispetto a quello utilizzato per la gestione dell'inizializzazione delle chain; la separazione è stata attuata in quanto i due progetti hanno obiettivi differenti. Il repository relativo agli attacchi, che saranno discussi nelle sezioni a seguire, è pubblicamente disponibile presso: `https://github.com/NorwegianGoat/bc_connector`.

- l'utente effettua un deposito utilizzando un bridge sulla sua catena di origine, il bridge è regolarmente funzionante e blocca i fondi dell'utente con successo, generando quindi un evento di deposito;
- l'utilizzatore del bridge si muove verso la catena di destinazione e, una volta arrivato, riconfigura il proprio relayer per trasmettere il deposito al bridge di riferimento per quella blockchain;
- il bridge della chain di destinazione è però configurato per puntare a un contratto malevolo, che è in tutto e per tutto aderente allo standard ERC-20 o ERC-721, ma che utilizza una funzione `mint` in cui l'indirizzo beneficario dei token è hardcoded, come riportato nel listato 3.1;
- i token depositati nella nuova chain vengono quindi creati per un utente che non è quello che ha effettivamente avviato la proceduta di deposito.

```
address evilAddress = address(0
    xD9635866Ade8E73Cc8565921F7CF95f5Be8f6D3e);

function mint(address to, uint256 amount) public virtual override {
    require(
        hasRole(MINTER_ROLE, _msgSender()),
        "ERC20PresetMinterPauser: must have minter role to mint"
    );
    _mint(evilAddress, amount);
}
```

Listing 3.1: Funzione di mint modificata ERC-20 malevolo.

Nel diagramma di sequenza della figura 3.1 è rappresentato quanto descritto fino ad ora.

L'attacco appena presentato, oltre al furto dei token sulla catena di destinazione, ha un ulteriore risvolto da tenere in considerazione, ossia che l'indirizzo che ha sottratto i token può ora disporne come preferisce, ad esempio anche depositando i fondi su un'altra chain, impedendo di fatto che l'utente possa pretendere di riottenerli con una normale transazione di rimborso nella blockchain in cui si trova.

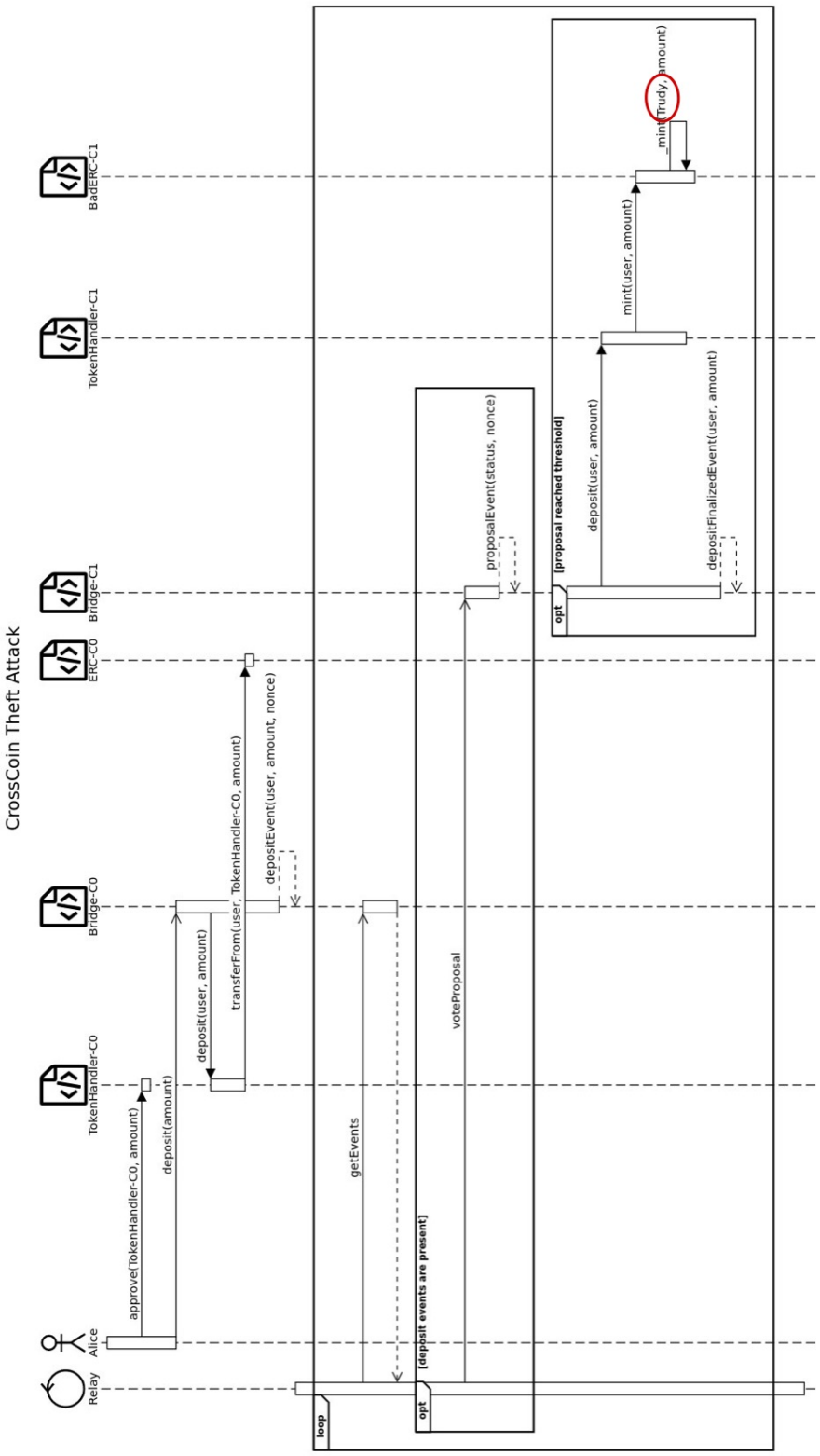


Figura 3.1: Diagramma di sequenza CrossCoin Theft Attack

3.1.1 Requisiti per la protezione dall'attacco

3.1.1.1 Separazione delle transazioni

Una delle modalità con cui può essere mitigato il CrossCoin Theft Attack consiste nella separazione del processo di deposito dei token in due transazioni distinte: con una prima transazione l'utente invia un piccolo numero di token al fine di verificare che questi vengano effettivamente creati sulla blockchain di destinazione a suo nome, in seguito, qualora la prima transazione fosse andata a buon fine, esso procederebbe con una seconda transazione, riguardante la restante parte dei token da trasferire.
L'approccio appena descritto presenta tuttavia alcune limitazioni. Nel caso in cui il deposito riguardasse token non fungibili, a causa della loro unicità, il furto di anche uno solo dei token rappresenterebbe un evento negativo.

Un ulteriore aspetto da tenere in considerazione riguarda la fattibilità del processo di split delle transazioni. Come già descritto in precedenza, l'utente, effettuato il primo deposito, si muove dalla chain di origine a quella di destinazione con un nodo ed il relayer; qualora l'utilizzatore del bridge volesse generare la seconda transazione, sarebbe necessario trasmettere la nuova transazione alla chain di origine, che però non sarebbe raggiungibile. L'utente dovrebbe quindi recarsi in un luogo da cui poter contattare la blockchain di origine per effettuare il secondo deposito e tornare nuovamente nel range della chain di destinazione per trasmettere la seconda transazione. È chiaro che, in scenari in cui le distanze da coprire fossero considerevoli, quanto appena descritto rappresenterebbe un notevole ostacolo in termini di tempo e risorse; inoltre nulla garantisce che il bridge non possa subire alterazioni nel tempo che intercorre tra il primo deposito e il secondo.

3.1.1.2 Verifica bytecode smart contract

Un'ulteriore possibile soluzione all'attacco è rappresentata da una verifica anticipata del bytecode dei contratti con cui l'utente andrà ad interagire. Disponendo del codice sorgente degli smart contract, è possibile effettuarne la compilazione per confrontare il bytecode risultante con quello del contratto del token per cui il bridge è impostato. Per ottenere il bytecode dello smart contract in esecuzione sulla chain di destinazione è

sufficiente effettuare una chiamata JSON-RPC col metodo `eth_getCode` a un nodo della catena; la verifica tra bytecode compilato e quello in esecuzione dalla blockchain consiste poi in un semplice confronto.

Anche l'approccio appena descritto presenta tuttavia delle limitazioni: la verifica di corrispondenza tra bytecode in esecuzione sulla blockchain di destinazione e quello generato dalla compilazione dello smart contract ha senso solo se si reputa sicuro il codice sorgente dello smart contract, in caso contrario l'utente decide di non interagire con il contratto e, pertanto, non procede a verificarne la corrispondenza. Per far ciò è richiesto che l'utente sia in grado di leggere il codice sorgente dello smart contract e di valutare se questo presenti delle irregolarità[3]; tale operazione può richiedere un quantitativo di tempo dipendende dalla lunghezza e complessità del contratto, tuttavia questa complicazione potrebbe essere mitigata mediante un controllo automatizzato, ad esempio mediante una suite di test, volto a verificare il corretto funzionamento del contratto. La seconda problematica che si riscontra è identica a quella che emerge dallo split della transazione di deposito: qualora l'utente verificasse che lo smart contract ha un comportamento corretto, e che il bytecode sulla blockchain di destinazione corrisponde a quello da lui compilato, sarebbe necessario tornare in prossimità della catena di origine per avviare il processo di deposito.

3.1.1.3 Gestione del bridge da parte di un'autorità esterna

Una migliore soluzione al problema dell'interazione con contratti potenzialmente non fidati è rappresentata dal delegare il deploy e la gestione degli smart contract a un'autorità o un ente esterno, privo di interessi nelle blockchain di origine e di destinazione. La gestione dei bridge da parte di un'autorità esterna comporta che questa abbia obiettivi

[3]L'utente svolge fondamentalmente una operazione di audit della sicurezza degli smart contract con cui si troverà a interagire. L'interazione con smart contract deve essere chiaramente consapevole da parte degli utenti; casi di furti di fondi ad opera di smart contract malevoli saranno destinati ad essere più frequenti al crescere dell'adozione della blockchain, come accaduto per gli attacchi informatici, sempre più frequenti a seguito della generale informatizzazione della società. Tuttavia è impensabile che tutti gli utilizzatori di smart contract possano disporre di capacità tali per cui possano valutare la sicurezza dei contratti con cui interagiranno. A partire da queste premesse è probabile che si assisterà alla nascita di aziende specializzate nella certificazione e valutazione della sicurezza di smart contract.

diversi rispetto a un individuo che opera all'interno della blockchain per cui ha istanziato un bridge: nello specifico, l'interesse di "lungo periodo" dell'ente che gestisce la connessione tra le chain consiste nell'incassare le fee di utilizzo del bridge e per farlo deve garantire il corretto funzionamento della piattaforma e la sua sicurezza di utilizzo.

3.1.1.4 Amministrazione distribuita

La gestione di un bridge da parte di agenti differenti dagli utilizzatori garantisce che non vi siano interessi conflittuali all'interno delle chain (e.g. furto di fondi di altri utenti, etc.); tuttavia l'approccio descritto non comporta che un'organizzazione, seppur fidata e autorevole, non possa essere vittima di un furto delle proprie chiavi private[4]. Attualmente i contratti di ChainBridge prevedono che il Bridge Admin sia rappresentato da una singola entità, che ha il controllo totale sulle impostazioni del bridge; tuttavia, modificando gli smart contract che ne definiscono il funzionamento, è possibile implementare un sistema per cui alcune azioni di amministrazione del bridge richiedano il raggiungimento di uno specifico threshold di voti prima di essere eseguite. Ad esempio, supponendo che un attore malevolo volesse rimappare verso uno smart contract malevolo un bridge correttamente funzionante[5], basterebbe che esso venisse in possesso della chiave privata dell'organizzazione che gestisce il bridge. Qualora invece ci trovassimo nel contesto multi admin ipotizzato in precedenza, sarebbe richiesto il furto di k chiavi, con k corrispondente al threshold di voti necessari per effettuare una modifica al bridge. Per evitare situazioni di collusione tra admin (i.e. k admin che si accordano per riconfigurare un bridge) può essere richiesto loro di effettuare un versamento di una determinata quantità di fondi per poter assumere tale ruolo; in caso di comportamento scorretto i fondi non verranno restituiti e ciò fungerebbe da disincentivo ad intraprendere azioni malevole.

È difficile stabilire con certezza il numero di admin necessari per rendere l'amministrazione del bridge sufficientemente distribuita e la quantità di fondi da versare per

[4]Un possibile scenario è quello di un dipendente che ruba la chiave privata di gestione del bridge per utilizzarla a suo favore, ma un attacco analogo potrebbe anche provenire da fonti esterne; un esempio è quanto recentemente avvenuto con il bridge Ronin. Per ulteriori informazioni si rimanda a: https://www.bloomberg.com/news/articles/2022-03-29/hackers-steal-590-million-from-ronin+.

[5]Quanto appena descritto può essere svolto fornendo alla funzione `registerResource` lo stesso `resourceId` precedente e i nuovi indirizzi per gli smart contract da associare alla risorsa.

garantire che non verranno attuati comportamenti malevoli, in quanto ciò dipende dalla determinazione e dalle risorse a disposizione dell'attaccante; tuttavia un numero di admin maggiore o uguale a 10 ed un threshold minimo di voti impostato al 60% del totale degli amministratori può rappresentare una base di partenza sufficiente.

Per quanto riguarda il quantitativo di fondi da far depositare agli amministratori a garanzia del loro comportamento, esso dipende dal valore della valuta di scambio sulla blockchain; tuttavia, ipotizzando di utilizzare dei wrapped token[6] 1 wETH[7] può svolgere la funzione di deterrente all'attuazione di condotte scorrette. La condizione appena descritta può essere resa ulteriormente restrittiva rendendo la retribuzione degli admin periodica (e.g. trimestrale, semestrale o annuale) al fine di disincentivare ulteriormente comportamenti scorretti.

3.2 Fake Lock Attack

Il Fake Lock Attack rientra nella categoria del double spending, in quanto l'utente è in grado di generare un numero di depositi illimitato riguardanti gli stessi token.

L'attacco richiede che il `TokenHandler`[8] associato al bridge sia modificato in modo tale che la funzione di deposito non blocchi realmente i token dell'utente; un esempio di ciò è ben visibile nel listato 3.2 alle righe 5 e 7.

```
1 function deposit(bytes32 resourceID,address depositer,bytes calldata
    data) external override onlyBridge returns (bytes memory) {
2     ...
3     // No token gets burned or locked.
4     if (_burnList[tokenAddress]) {
5         burnERC20(tokenAddress, depositer, 0);
```

[6] "Un wrapped token è un token che rappresenta una cryptocurrency da un'altra blockchain [...] e ha lo stesso valore della cryptovaluta originale. A differenza della cryptocurrency originale, il wrapped token può essere utilizzato su certe blockchain non-native e successivamente scambiato per la cryptovaluta originale". La definizione è di R. Stevens. *What Are Wrapped Tokens.* 2022. Coindesk.

[7] Con wETH ci si riferisce a wrapped Ethereum, ossia la versione ERC-20 di Ethereum.

[8] Il `TokenHandler` è la classe base a partire dalla quale vengono implementati l'`ERC20Handler` e l'`ERC721Handler`. In questo capitolo si utilizzerà questa classe durante la discussione delle vulnerabilità, in quanto queste prescindono dal tipo di token che viene trasferito tra le blockchain.

```
6       } else {
7           lockERC20(tokenAddress, depositer, address(this), 0);
8       }
9   }
```

Listing 3.2: Funzione di deposito `TokenHandler` modificata.

In sostanza l'amministratore del bridge effettua il deploy di uno smart contract il cui codice sorgente è stato modificato rispetto alla versione originale dei contratti distribuiti da ChainBridge.

L'attacco, descritto dal diagramma di sequenza 3.2, è così riassumibile:

- l'utente genera un deposito di token per i quali il `TokenHandler` non effettua il blocco o la distruzione, il bridge emette comunque correttamente l'evento di deposito; l'utente può quindi continuare a effettuare depositi, per i quali non verranno bloccati i suoi fondi, limitandosi a pagare il gas necessario per eseguire le chiamate agli smart contract del bridge;

- una volta che l'utilizzatore del bridge ha effettuato il numero di depositi desiderato, esso si reca presso la blockchain di destinazione e trasmette gli eventi di avvenuto deposito;

- il bridge di destinazione processa i dati che vengono inviati e accredita all'utente un totale di token pari alla somma di quelli interessati dai trasferimenti multipli.

Oltre al danno arrecato alle chain di destinazione, all'interno delle quali gli utenti possono usare i loro token per acquistare beni e servizi a costo zero, bridge che permettono di muovere token senza bloccarli rappresentano un rischio per l'intero sistema; infatti, qualora alcuni utenti venissero a conoscenza della possibilità offerta da queste blockchain, potrebbero muovere token dalla loro chain di origine, bloccandoli regolarmente, salvo poi effettuare depositi illimitati dalla blockchain di destinazione verso quella di origine, ottenendo, senza destare sospetti[9], più token di quelli di partenza.

[9]Quanto descritto si tratta sostanzialmente di una "triangolazione" di fondi, la cui creazione, a causa della distanza tra blockchain, difficilmente può essere verificata e contestata, se non analizzando il traffico delle transazioni.

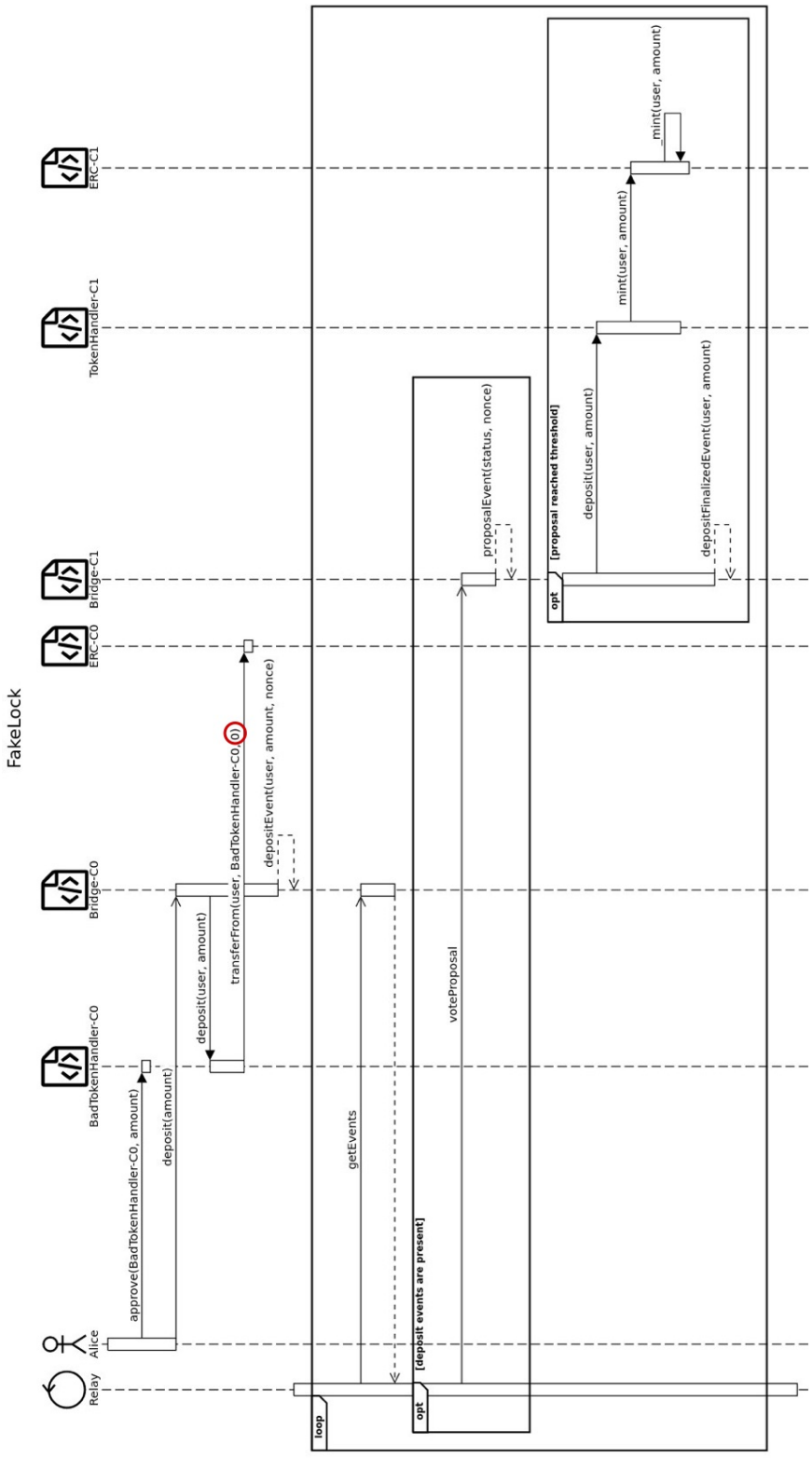


Figura 3.2: Diagramma di sequenza Fake Lock Attack

3.2.1 Requisiti per la protezione dall'attacco

3.2.1.1 Gestione del bridge da parte di un'autorità esterna

Il Fake Lock Attack si basa su un presupposto identico a quello del CrossCoin Theft Attack, ossia che uno dei contratti con cui l'utente interagisce sia malevolo; ciò che cambia rispetto all'attacco precedente è che i danni non si ripercuotono sull'utente, ma sulla blockchain con cui esso interagisce.

La motivazioni che spingono un bridge admin a istanziare un token handler che non effettua il lock o il burn dei fondi degli utenti possono derivare dal desiderio di porre la sua blockchain in una posizione di vantaggio rispetto alle altre (e.g. acquistare beni e servizi gratuitamente presso altri luoghi).

3.2.1.2 Riconfigurazione condizionale relayer

Esternalizzando l'istanziazione del bridge, la sua verifica di corretto funzionamento, manutenzione e gestione si eliminano gli interessi di un amministratore all'interno della chain che si interfaccia con le altre; tuttavia questa soluzione non garantisce che il relayer trasmetterà eventi letti da smart contract effettivamente gestiti da un'organizzazione terza, come quella teorizzata nella sezione 3.1.1.3.

Supponendo che il bridge admin autorizzi un utente a trasmettere eventi dalla sua blockchain di origine solo se il bytecode dei contratti monitorati dal relayer coincidesse con quello dei contratti sulla blockchain di destinazione, il controllo verrebbe effettuato sugli indirizzi degli smart contract per cui l'utente dichiara di aver configurato il relayer. Il bridge admin non ha però la certezza che il bytecode dei contratti che verificherà sia realmente quello di cui il relayer effettua il monitoraggio, in quanto solamente il proprietario del relay ha il controllo sulla sua configurazione.

In uno scenario in cui è presente un numero di relay superiore a uno, la certezza che l'evento di deposito sia realmente avvenuto si ottiene mediante i voti dei vari relayer, configurati per puntare tutti allo stesso set di contratti. Qualora un singolo relayer trasmettesse un evento non veritiero, il threshold di voti necessari alla esecuzione del deposito non verrebbe raggiunto e quindi nessun fondo verrebbe accreditato; tuttavia, a causa delle caratteristiche dello scenario d'uso ipotizzato nella sezione 2.3, non vi è la

possibilità di disporre di più relayer che monitorino gli stessi contratti al fine di votare l'effettiva esecuzione di un deposito.

Una protezione migliore dall'eventualità di trasmissione di eventi da contratti differenti rispetto a quelli verificati dal bridge admin può essere offerta qualora il relay fosse gestito direttamente dalla blockchain di destinazione. In questo caso un individuo che volesse effettuare un deposito da smart contract malevoli dovrebbe fornire l'indirizzo del bridge realmente utilizzato, in quanto altrimenti non verrebbero letti gli eventi relativi alle transazioni da trasmettere. Forniti gli indirizzi dei contratti che saranno interessati dalla lettura degli eventi, il gestore del relayer verificherà tali contratti e procederà a riconfigurare il software di relay solamente se il bytecode dei contratti risulterà identico a quello certificato dall'organizzazione che ha istanziato il bridge.

In definitiva, il Fake Lock attack è risolvibile con i seguenti accorgimenti: deploy dei contratti del bridge da parte di un'autorità conosciuta, che garantisce il corretto funzionamento degli smart contract, e controllo del relay affidato alla blockchain di destinazione, che effettua la riconfigurazione del software solo a seguito di operazioni auditing degli smart contract che verranno monitorati.

3.3 Token Drain Attack

Il Token Drain Attack è un attacco che rientra nella categoria del fund stealing; tuttavia, a differenza di quanto descritto nella sezione 3.1, l'appropriazione di token di altri utenti non avviene mediante furto nel processo di deposito, ma effettuando dei depositi da altre chain.

L'attacco ha il seguente andamento:

- supponendo che un utente si trovi in una catena dove è presente un bridge che punta ad un contratto in cui il mint dei token non è limitato in alcun modo, l'utente può creare un quantitativo di token arbitrario e inviarli alla chain di destinazione;
- la catena che riceve il deposito fornisce all'utente un quantitativo di token pari a quello depositato, prelevandoli dai token nel bilancio del `TokenHandler`, in caso di utilizzo del meccanismo lock-unlock, o creandoli, qualora venisse usato il meccanismo burn-mint.

Qualora ci trovassimo nel primo caso descritto, ossia di una blockchain con lock-unlock dei fondi, ad ogni token inviato in ingresso ne corrisponderebbe uno prelevato da quelli che altri utenti hanno depositato in uscita dalla chain; di conseguenza l'utente che ha effettuato il deposito si approprierebbe di fondi la cui creazione è stata legittima (e.g. gli utenti hanno acquistato i token che hanno depositato in uscita inviando un controvalore nella valuta della chain locale), a fronte di un invio di token il cui controvalore è pari a zero, essendo stati generati su una catena che non impone limitazioni.

A differenza di depositi tutto sommato limitati, come possono essere i depositi riguardanti token creati regolarmente, per i quali sono state barattate risorse scarse (i.e. valuta della chain locale), i depositi di fondi creati senza limitazioni possono riguardare cifre consistenti o, qualora l'attaccante volesse passare inosservato, svariati piccoli depositi; quanto descritto potrebbe causare il blocco del bridge per le transazioni in ingresso, in quanto il bilancio del `TokenHandler` potrebbe essere azzerato per far fronte all'enorme quantitativo di token depositati in entrata. L'effetto appena descritto può avvenire anche a seguito del Fake Lock Attack: in fin dei conti generare depositi illimitati per gli stessi token o poter generare token senza limitazioni e in seguito depositarli porta alle stesse conseguenze.

3.3.1 Requisiti per la protezione dall'attacco

3.3.1.1 Distruzione dei token

La complessità di mitigazione dell'attacco appena presentato dipende dal tipo di bridge che si vuole istanziare. Qualora fosse ritenuto importante mantenere invariato il quantitativo di token totali in circolazione nel sistema, sarebbe necessario lasciare il bridge impostato nella modalità di deploy di default, ossia lock-unlock; al contrario, qualora il quantitativo di token circolanti in una blockchain fosse irrilevante, si potrebbe optare per la modalità burn-mint.

Impostando la modalità burn-mint il problema del Token Drain Attack non si pone: ogni token che viene trasferito in ingresso viene infatti creato *ex-novo*, conseguentemente è impossibile che il bridge rimanga bloccato in quanto il `TokenHandler` non dispone di fondi nel suo bilancio.

3.3.1.2 Supply controllata tra le diverse chain

La modalità lock-unlock può essere resa sicura qualora nelle chain di origine dei depositi fossero presenti smart contract che rendono impossibile il mint illimitato di token; inoltre è obbligatorio che la supply totale dei token su ogni catena sia identica[10].

La soluzione appena presentata comporta tuttavia una problematica di natura logistica: qualora la quantità totale dei token per cui viene creato il bridge dovesse cambiare, si renderebbe necessario trasmettere tale cambiamento a tutte le chain di destinazione. Questa trasmissione potrebbe essere svolta mediante un sistema di messaggi gestito attraverso un `GenericHandler`; tuttavia, considerando che il cambio di supply potrebbe accadere per ciascun token del bridge e che il tempo di propagazione del messaggio verso le blockchain di destinazione varia in funzione della distanza tra le catene e del numero di persone che vi si recano, la soluzione mediante burn-mint dei token sembra maggiormente robusta e, pertanto, più appropriata.

3.4 Admin Corruption Attack

L'Admin Corruption Attack rientra nella categoria del fund stealing, in quanto alcuni token vengono sottratti indebitamente da quelli conservati dal `TokenHandler`. Il diagramma di sequenza in figura 3.3 rappresenta le varie fasi dell'attacco.

Tra le funzionalità disponibili all'amministratore del bridge, oltre a quelle relative alla configurazione dei contratti, ve n'è una nominata `adminWithdraw`; tale funzione permette all'admin di prelevare fondi versati dagli utenti e di inviarli ad un indirizzo a sua scelta. La funzione appena descritta è riportata nel listato 3.3.

```
function adminWithdraw(address handlerAddress, bytes memory data)
    external onlyAdmin {
    IERCHandler handler = IERCHandler(handlerAddress);
    handler.withdraw(data);
}
```

Listing 3.3: Funzione per il prelievo dei fondi da parte dell'admin.

[10] L'ultimo punto risulta necessario per coprire casi limite come, ad esempio, quello in cui tutti i token venissero inviati verso un unica blockchain.

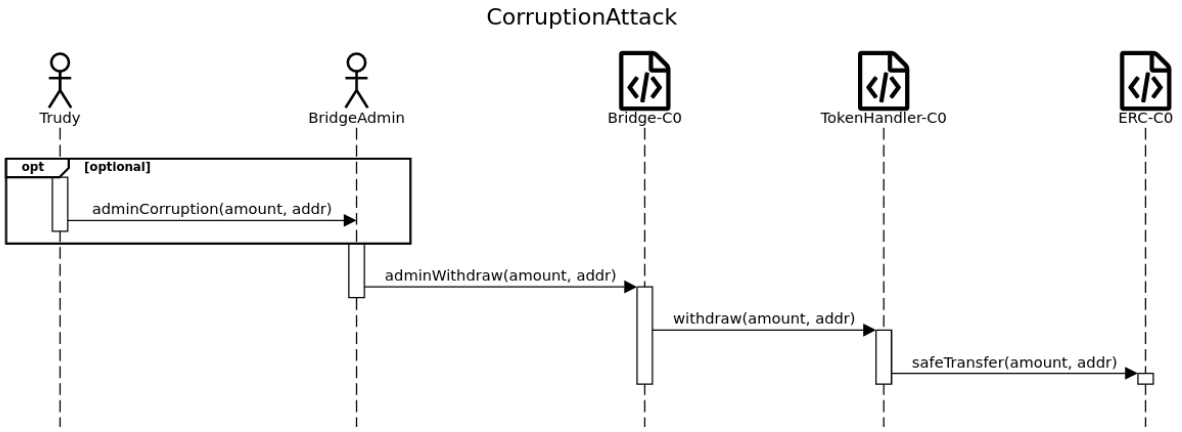


Figura 3.3: Diagramma di sequenza Admin Corruption Attack

È interessante evidenziare come la chiamata alla funzione `withdraw` dell'handler non effettui alcun controllo sui dati passati dal bridge admin, ma consista in una semplice chiamata alla funzione `safeTransferFrom` del contratto ERC-20 o ERC-721 per cui è mappato il bridge.

L'admin è quindi potenzialmente libero di appropriarsi dei fondi trasferiti dagli utenti in uscita dal bridge e di inviarli a chiunque senza alcuna limitazione. La funzione di `adminWithdraw`, da questo punto di vista, potrebbe sembrare una sorta di backdoor all'interno degli smart contract, tuttavia tale funzionalità può risultare utile qualora un deposito di un utente non andasse a buon fine sulla chain di destinazione e i fondi rimanessero bloccati nel `TokenHandler` di origine. La situazione appena descritta può originarsi qualora siano passati troppi blocchi dal deposit proposal ad opera del relayer; ciò avviene qualora lo smart contract del bridge di destinazione fosse configurato per cancellare un deposito che non abbia ricevuto un numero di voti minimi pari al threshold dopo un determinato numero di blocchi. In un caso del genere, l'utente rimane privato dei suoi fondi sulla chain di origine, in quanto bloccati a inizio deposito, senza che il controvalore gli venga accreditato sulla catena di destinazione, pertanto la funzionalità di rimborso è necessaria per evitare che i fondi dell'utente vengano persi.

Come accennato in precedenza, il bridge admin potrebbe però fare un uso scorretto di questa funzionalità in seguito alla corruzione da parte di utenti che, in cambio di un invio di token prelevati indebitamente dal `TokenHandler`, garantiscono all'amministratore un

qualche genere di corrispettivo per l'azione malevola. In alternativa, l'admin potrebbe essere portato a prelevare fondi dopo essere stato tratto in inganno, ad esempio, qualora un utente affermasse che il deposito non è andato a buon fine sulla chain di destinazione poichè il numero di relayer richiesti per la votazione del deposit proposal è stato impostato a più di uno.

3.4.1 Requisiti per la protezione dall'attacco

3.4.1.1 Amministrazione distribuita e rendicontazione operazioni

Il principio su cui si basa l'Admin Corruption Attack consiste nella centralizzazione dei diritti di amministrazione del bridge.

L'attacco descritto precedentemente riguarda il furto di fondi da parte di un amministratore corrotto; tuttavia la gamma di azioni malevoli intentabili da un singolo amministratore è varia e, come già accennato nella sezione 3.1.1.4, può comprendere la modifica di bridge funzionanti, che vengono rimappati verso smart contract malevoli, ma anche la messa in pausa di un bridge, con conseguente blocco dei token degli utenti all'interno della blockchain e delle transazioni in ingresso, etc.

Per evitare furti di token come quello descritto nella sezione precedente, la soluzione più semplice consiste nel far rendicontare al bridge admin il motivo del rimborso di token e di legare la sua retribuzione al corretto svolgimento delle sue mansioni; tuttavia risulta evidente che una semplice rendicontazione potrebbe scoraggiare, ma non impedire azioni scorrette, specialmente se il profitto originato da tali azioni supera il compenso dell'amministratore.

3.4.1.2 Distruzione token

Una possibile soluzione alternativa consiste nell'impostazione di ogni bridge in modalità burn-mint, similmente a quanto descritto nella sezione 3.3.1.1. Come visibile nel diagramma di sequenza della figura 3.3, l'ultima chiamata degli smart contract consiste nell'esecuzione della funzione `safeTransferFrom` da parte del `TokenHandler`. Qualora il bridge fosse configurato in modalità mint-burn, i token spostati verso un'altra blockchain non rientrerebbero mai nel bilancio del `TokenHandler`, in quanto verrebbero istantanea-

mente distrutti; questo renderenderebbe impossibile per il bridge admin trasferire una certa quantità di token a terzi prelevandoli dal `TokenHandler`. L'esecuzione di tale operazione, in un contesto dove la modalità di blocco dei fondi fosse la burn-mint, produrrebbe infatti un errore al raggiungimento della quarta riga del listato 3.4, in quanto il bilancio del gestore dei token sarebbe 0 e pertanto non disporrebbe dei token da inviare, su comando dell'admin, ad altri utenti.

```
1 function releaseERC20(address tokenAddress, address recipient, uint256
    amount) internal {
2         IERC20 erc20 = IERC20(tokenAddress);
3         _safeTransfer(erc20, recipient, amount);
4 }
```

Listing 3.4: Erc20Safe.sol release.

3.4.1.3 Aggiunta controllo a livello di smart contract sui token rimborsati

Qualora l'admin del bridge volesse comunque mantenere utilizzabile la modalità lock-unlock, uno schema multi admin, come quello suggerito nella sezione 3.1.1.4, consentirebbe di evitare decisioni arbitrarie di amministratori e comporterebbe la necessità di corrompere un numero di bridge admin pari al numero di voti necessari al raggiungimento della soglia minima impostata durante la creazione del bridge. Inoltre, la funzione di withdraw potrebbe essere modificata in modo tale da permettere il rimborso solamente di un numero di token pari a quello depositato; in questo modo si eliminerebbe anche la possibilità di commettere errori nel corso della procedura di rimborso.

3.5 Rollback Attack

Il Rollback Attack rientra tra gli attacchi double spending, in quanto, anche in questo caso, vengono effettuati depositi multipli riguardanti gli stessi token.

L'attacco prevede che l'utente che vuole svolgere il deposito si accordi con gli utenti che controllano i nodi della blockchain per ripristinare lo stato della chain ad un momento immediatamente precedente al trasferimento dei token, obiettivo che può essere raggiunto effettuando un backup della cartella `data` del Polygon-Edge SDK: all'interno

di tale directory vengono infatti memorizzate tutte le strutture dati della blockchain, che pertanto potranno essere ripristinate mediante il backup.

Una volta che i nodi hanno effettuato il backup, l'utente effettua il deposito e si muove verso la blockchain di destinazione per trasmettere il deposit event, nel frattempo il backup dello stato precedente della chain viene ripristinato e, conseguentemente, l'utente potrà generare nuovi depositi per gli stessi token.

3.5.1 Requisiti per la protezione dall'attacco

3.5.1.1 Aumento numero di nodi partecipanti alle chain locali

L'attacco presentato nella sezione precedente è probabilmente uno dei più complessi da risolvere in quanto si basa sulla scarsa distribuzione delle blockchain nelle zone non raggiunte dalla connettività. Una possibile soluzione alla scarsa decentralizzazione potrebbe consistere nell'incremento del numero di nodi delle chain, ad esempio, facendo gestire un nodo ad individui di altre blockchain; tuttavia ciò comporterebbe in un notevole dispendio di hardware a fronte di un controllo non diretto dei nodi, che, per le caratteristiche dello scenario ipotizzato, sarebbero irraggiungibili in quanto distanti dalle blockchain di origine dei proprietari.

3.5.1.2 Utilizzo di Merkle proofs e configurazione condizionale relayer

Il rollback attack può trovare soluzione mediante l'utilizzo dei Merkle trie per la verifica dello stato della blockchain da cui si sta ricevendo l'evento di deposito.

I Merkle trie sono strutture dati ad albero che consentono di verificare l'integrità dei valori immagazzinati in tali strutture. I valori assunti da un trie sono completamente deterministici, in quanto si basano sull'hashing dei blocchi di dati contenuti nell'albero; ad esempio, nel caso di un Merkle trie binario, ad ogni livello viene effettuato l'hash di due chunk di dati a partire dalle foglie, che contengono i valori di cui si vuole verificare l'integrità e l'inclusione, fino ad arrivare ad avere una root, che è il prodotto dell'hashing

di tutti i livelli sottostanti. Due trie formati dagli stessi chunk di dati di partenza produrranno quindi una struttura ad albero identica[11].

Le chain basate su Ethereum Virtual Machine dispongono di quattro trie fondamentali:

- state trie: contiene le informazioni di stato di un account, sia esso un externally owned account, ossia un account controllato da un utente, o uno smart contract. Le informazioni riguardanti l'account comprendono: il nonce, il bilancio, lo storage root[12] e il code hash, ossia l'hash del codice dell'account nella EVM[13];

- storage trie: contiene i dati associati alle variabili di un determinato account. Gli externally owned account, non disponendo di variabili, non hanno uno storage trie;

- transaction trie: è un albero che contiene le informazioni relative alle transazioni incluse in un blocco, ossia: nonce della transazione, importo, gas price, campo data[14], etc. A differenza dei due alberi discussi precedentemente, il transaction trie di un blocco è immutabile in quanto le transazioni incluse in un blocco non possono cambiare;

- receipts trie: contiene le informazioni relative all'esecuzione di tutte le transazioni della chain. È possibile pensare il transaction trie e il recipits trie come complementari: la prima struttura dati descrive gli input di una transazione inseriti in un blocco, mentre la seconda gli effetti. Tra le informazioni contenute nel receipts trie vi sono: hash del blocco in cui è stata inclusa la transazione, numero del blocco, hash della transazione, mittente, destinatario, gas effettivamente utilizzato, log generati, etc.

La verifica via Merkle trie dello stato della chain da cui si ricevono eventi di deposito prenderebbe in considerazione il contenuto dello storage trie relativo allo smart contract

[11]Per una completa disamina sul funzionamento e sull'utilizzo di questa struttura dati si rimanda a `https://eth.wiki/en/fundamentals/patricia-tree` e `https://arxiv.org/pdf/2108.05513.pdf`.

[12]Lo storage root è la radice di un altro Merkle trie relativo all'account in questione.

[13]Nel caso di un externally owned account, non essendo in esso presente codice da eseguire, il code hash è l'hash di una stringa vuota.

[14]Il campo data contiene i parametri in input in caso la transazione riguardasse l'interazione con uno smart contract.

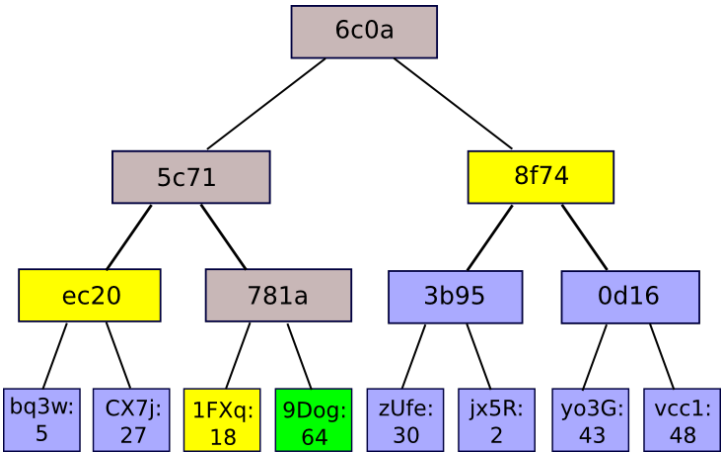


Figura 3.4: Esempio di Merkle Trie. Immagine tratta dall'Ethereum fundation blog.

del bridge. Considerando che ogni deposito di ChainBridge contiene un nonce, un secondo deposito, a seguito del ripristino dei trie da parte di nodi melevoli, dovrebbe avvenire utilizzando un nonce differente; al contrario il bridge di destinazione rigetterebbe il nuovo deposito in quanto il nonce risulterebbe già utilizzato. Supponendo dunque l'invio dello stesso deposito iniziale, ma con nonce differente, la root dello storage trie relativo al bridge della chain di origine cambierebbe.

L'utilizzo di una Merkle proof per verificare che un determinato set di dati sia effettivamente ancora parte dello storage trie[15] consiste nel calcolare la root di un Merkle trie a partire da un sottoinsieme di dati.

Supponendo che l'albero in figura 3.4 rappresenti lo storage trie di un contratto bridge, il bridge admin può verificare che una blockchain da cui stanno venendo depositati fondi non abbia effettuato il Rollback Attack nel seguente modo:

1. viene recuperata la Merkle proof per l'ultimo deposito che è stato ricevuto dalla blockchain di origine. La Merkle proof, o i dati per costruirla (i.e. i dati all'interno dello storage del contratto), dovrebbe essere stata salvata in precedenza sulla blockchain di destinazione o comunque in uno spazio in cui ne è garantita l'immutabilià

[15]Un esempio di processo di verifica di una storage proof è descritto presso `https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/`.

e la disponibilità, come, ad esempio IPFS[16]. In alternativa, è possibile scrivere su blockchain anche solo la root del Merkle trie e utilizzare questa informazione per verificare che il ripristino del trie da cui ottenere la Merkle proof sia andato a buon fine, a prescindere da dove i dati siano stati salvati in precedenza. La Merkle proof consiste nella root del Trie da verificare, negli hash delle root dei sottoalberi che portano fino al chunk di dati[17] di cui si vuole provare l'inclusione e nel blocco di dati stesso;

2. a partire dal blocco di dati da verificare (blocco verde nella figura) e dagli hash delle radici dei sottoalberi (blocchi gialli) vengono calcolati i nuovi hash (blocchi grigi) risalendo fino alla root dello storage trie;

3. se la root dello storage trie appena calcolata è identica a quella fornita insieme alla Merkle proof, allora è possibile affermare che non vi è stato alcun rollback dello stato della blockchain; al contrario, se la root calcolata differisce da quella fornita con la Merkle proof, ciò significa che lo storage trie ha subito delle modifiche (i.e. è stato effettuato un rollback di stato a cui è seguito un deposito differente).

In definitiva, se la Merkle proof a disposizione della chain di destinazione è compatibile con l'attuale storage trie relativo bridge di origine, l'utente proviene da una chain che non ha effettuato dei rollback di stato e pertanto il relayer verrà riconfigurato per la trasmissione degli eventi su questo bridge; di contro, qualora la merkle proof per l'ultimo deposito ricevuto, conservata dalla chain di destinazione, risultasse incompatibile con l'attuale storage trie del bridge contract della catena di origine del deposito, il relayer non verrebbe riconfigurato.

3.6 ChainId Collision Attack

Il ChainId Collision Attack rientra nella tipologia degli attacchi bridge poisoning, in quanto il relayer trasmette eventi relativi a una blockchain con un ChainId identico a

[16]Per ulteriori informazioni su IPFS si rimanda a: `https://docs.ipfs.io/concepts/what-is-ipfs/#decentralization`.

[17]Nel caso in analisi si fa riferimento ai dati relativi a un singolo evento di deposito.

quello per cui è stato configurato un determinato bridge, senza che tali eventi provengano però dalla chain a cui quel ChainId appartiene realmente.

Come è possibile notare dalla riga 5 del listato 3.5, il ChainId della blockchain a cui fa riferimento un deposito viene passato in input da parte del relayer che trasmette l'evento. Qualora il relayer appartenesse a un utente malevolo il cui obiettivo fosse quello di creare depositi per fondi che non possiede, quest'ultimo potrebbe istanziare una blockchain in cui sono presenti contratti che permettono il mint illimitato di token e, contestualmente, configurare un bridge per generare gli eventi di deposito.

Una volta generati i depositi, che non differiscono in nulla rispetto a un regolare deposito, se non per il fatto provengono da una catena creata *ad hoc*, questi vengono trasmessi e votati dal relayer sulla chain di destinazione, il cui contratto bridge accetterà il deposito in quanto proveniente da una blockchain conosciuta.

L'attacco appena descritto è possibile in quanto l'utente, avendo il possesso dell'unico relayer necessario per votare e far accettare al bridge il suo deposito, è l'unico garante dell'autenticità e della provenienza dei messaggi che vengono inviati agli smart contract.

```
1 function voteProposal(uint8 domainID, uint64 depositNonce, bytes32
    resourceID, bytes calldata data) external onlyRelayers whenNotPaused
    {
2     ...
3     if (proposal._status == ProposalStatus.Passed) {
4         executeProposal(domainID,depositNonce, data, resourceID, true);
5         return;
6     }
7     ...
8 }
```

Listing 3.5: Funzione per il voto dei depositi da parte dei relayer.

3.6.1 Requisiti per la protezione dall'attacco

3.6.1.1 Incremento del numero di relayer necessari per ultimare un deposito

L'attacco mediante collisione del ChainId può essere risolto attraverso l'utilizzo di più relayer, che votano la trasmissione di un evento; ad ogni modo, come nel caso del Rollback

Attack, le caratteristiche dello scenario ipotizzato nella sezione 2.3 non consentono di conoscere in anticipo se e quanti utenti provenienti da una stessa catena di origine si muoveranno verso una determinata chain di destinazione. Quindi imporre la presenza di più relayer rappresenterebbe una condizione eccessivamente restrittiva.

3.6.1.2 Utilizzo di Merkle proofs e configurazione condizionale relayer

La protezione da questo tipo di attacco passa per una soluzione tecnica analoga a quella presentata nella sezione 3.5.1.2.

Disponendo del Merkle trie relativo a tutti i trasferimenti passati provenienti da una specifica blockchain di origine, viene verificato che i precedenti depositi siano effettivamente presenti; qualora i trasferimenti passati differissero rispetto a quelli memorizzati nel Merkle trie a disposizione del brige admin, ciò significherebbe che l'utente che sta richiedendo l'autorizzazione a trasmettere depositi possiede sul suo nodo un bridge con uno storage trie contenente dati differenti da quelli inviati fino ad ora.

Qualora fosse identificata una differenza di contenuto di storage, il relayer non sarà riconfigurato per la lettura degli eventi dagli smart contract indicati dall'utente.

3.7 Raccolta dei requisiti

Nelle sezioni precedenti sono stati illustrati i requisiti per la protezione dai singoli attacchi; tali requisiti verranno ora raggruppati in una tabella riassuntiva e distinti in requisiti minimi e requisiti consigliati. I requisiti minimi rappresentano quegli accorgimenti che possono migliorare la sicurezza degli utenti del bridge e delle catene che ricevono trasferimenti, ma che comunque non risolvono totalmente i problemi individuati; al contrario, i requisiti consigliati rappresentano la soluzione giudicata più efficace per la mitigazione dell'attacco.

Attacco	**Requisiti Minimi**	**Requisiti Consigliati**

CrossCoin Theft	• Separazione deposito in due transazioni distinte. • Verifica anticipata del bytecode del contratto con cui l'utente interagirà.	• Gestione bridge da parte di un'autorità priva di interessi nelle chain di origine/destinazione. • Aggiunta di sistema di amministrazione distribuita. • Retribuzione admin bridge periodica e commitment fondi.
Fake Lock	• Gestione del bridge da parte di autorità esterna che effettua auditing periodico.	• Riconfigurazione del relayer solo se il bytecode dei contratti della blockchain di origine è coincidente con quelli della chain di destinazione.
Token Drain	• Modalità lock-unlock con audit dei contratti che gestiscono il mint dei token. • Supply totale dei token identica tra le varie catene.	• Impostazione bridge in modalità burn-mint.

Admin Corruption	• Rendicontazione azioni intraprese dall'admin sul bridge. • Retribuzione admin basata su rendicontazione delle azioni. • Commitment di fondi da parte degli amministratori e retribuzione periodica.	• Impostazione bridge in-modalità burn-mint. • Amministrazione distribuita con threshold di voti per svolgimento di determinate azioni. • Controllo sulla quantità di token rimborsata dagli admin agli utenti.
Rollback Attack	• Aumentare numero di nodi.	• Utilizzo di Merkle proofs per monitoraggio storage trie. • Riconfigurazione relayer per lettura eventi solo in caso di storage trie consistente rispetto alla Merkle proof.
ChainId Collision	• Aumentare il numero di relayer richiesti per il voto di un deposito.	• Verifica stato Merkle trie.

Come è possibile notare dal raggruppamento dei requisiti consigliati nella tabella precedente, gli attacchi individuati sono mitigabili utilizzando alcuni accorgimenti comuni: introduzione di admin multipli con imposizione di un threshold minimo di voti per lo svolgimento di determinate azioni, impostazione del bridge in modalità burn-mint, gestione del relayer da parte della catena di destinazione e non degli utenti che desiderano trasmettere i depositi, accettazione degli eventi in ingresso solo se lo storage trie del bridge di origine del deposito risulta essere compatibile con lo storage trie osservato in

precedenza e, infine, audit dei contratti da parte di un'organizzazione esterna, che ne garantisca il corretto funzionamento.

3.8 Sulla generalizzabilità degli attacchi

Gli attacchi descritti nel presente capitolo fanno riferimento a specifiche componenti dei contratti di ChainBridge e ad alcune assunzioni sullo scenario di utilizzo dei bridge; tuttavia è possibile individuare diverse caratteristiche di base che rendono generalizzabile quanto presentato anche per altri sistemi.

Il CrossCoin Theft Attack e il Fake Lock Attack sfruttano entrambi modifiche malevoli agli smart contract, grazie alle quali è possibile impossessarsi dei fondi degli utenti ed emettere eventi non veritieri. Contratti con errori di implementazione, o modifiche intenzionali, come nei casi citati, possono causare enormi perdite agli utilizzatori dei bridge, che fanno transitare i loro fondi attraverso gli smart contract. L'unica soluzione al problema della sicurezza e correttezza implementativa dei contratti è rappresentata dall'introduzione di un sistematico utilizzo di auditing del codice dei bridge. Nell'industria informatica tale pratica è ampiamente diffusa, sia tramite aziende specializzate, sia tramite programmi di bug bounty; tuttavia le aziende nel mondo della blockchain subiscono una forte pressione all'innovazione determinata dalla rapidità con cui il settore evolve e se ciò spinge gli sviluppatori ad aggiungere funzionalità per avere un vantaggio competitivo rispetto alla concorrenza, ciò avviene a discapito della sicurezza del codice prodotto[18].

Un'ulteriore problematica è quella della perdita della corrispondenza 1:1 del numero dei token che transitano tra le chain; tale meccanismo viene sfruttato attivamente nel Fake Lock Attack, nel Token Drain Attack, nel Rollback Attack e nell'Admin Corruption Attack. Un esempio di attacco condotto utilizzando questa tecnica è quello avvenuto ai danni del bridge Wormhole[19], in cui l'attaccante ha creato dei token WETH sulla block-

[18] È quanto osserva N. Shelby. C. Faife, *Explaining Crypto's Billion-Dollar Bridge Problem*, in The Verge, 2022.

[19] Per maggiori informazioni si rimanda a `https://www.cnbc.com/2022/02/02/320-million-stolen-from-wormhole-bridge-linking-solana-and-ethereum.html` e alla dettagliata analisi di CertiK `https://certik.medium.com/wormhole-bridge-exploit-analysis-5068d79cbb71`.

chain Solana per poi inviarli sulla chain di Ethereum. Pur essendo differenti le modalità tramite le quali si arriva alla perdita del peg dei token tra le chain (e.g. possibilità di mint illimitato dei token su una blockchain, attacchi 51% per annullare transazioni avvenute, rimborsi da parte dell'admin del bridge, etc.), quanto accomuna tutti questi attacchi è che "attaccare anche una sola chain può creare un contagio sistemico che mette in pericolo l'economia dell'intero ecosistema"[20]. Il problema è aggravato dal fatto che mantenere token all'interno dei bilanci degli smart contract dei bridge contribuisce ad aumentare il rischio che un bridge sia obiettivo di un attacco. In sostanza le attività cross-chain comportano una esternalità negativa: quando non c'è particolare attività su un bridge questo è abbastanza sicuro, ma al crescere delle transazioni che lo utilizzano è associato anche un maggiore rischio[21], derivante dalla quantità di fondi che rimangono depositati all'interno dei contratti. Sotto questo punto di vista l'utilizzo del meccanismo burn-mint contribuisce a rendere più sicuro il sistema in quanto, non rimanendo token all'interno dei contratti del bridge, l'attaccante dispone di un minor incentivo a commettere azioni malevole.

L'Admin Corruption Attack, il Rollback Attack e il ChainId Collision presentano invece un ulteriore profilo da tenere in considerazione, ossia l'eccessivo grado di centralizzazione di alcune componenti del sistema (i.e. amministrazione del bridge gestita da un singolo individuo, basso numero di relayer per la lettura degli eventi e nodi che formano la blockchain). Un esempio di attacco che in passato ha sfruttato questo aspetto è quello avvenuto ai danni del bridge Ronin: in questo caso l'attaccante ha "utilizzato chiavi private rubate[22] per creare prelievi falsi"[23]. Il problema dell'alto grado di centralizzazione nel Web3 non è nuovo ed anzi riprende una dinamica simile a quella che si è manifestata con il passaggio da Web1 a Web2, ossia con l'evoluzione da un Web di fruizione passiva dei contenuti, in cui gli utenti hostavano autonomamente le proprie risorse, al Web dina-

[20]V. Buterin, *[AMA] We are the EF's Research Team (Pt. 7: 07 January, 2022)*, 2022.

[21]Ibidem.

[22]Il caso del bridge Ronin è di particolare interesse in quanto l'attaccante ha fatto uso di "tecniche di social engeering [...] e ha sfruttato problemi di design della sicurezza invece che specifiche vulnerabilità software, come nella maggior parte degli altri hack di bridge. In particolare, altri attacchi hanno avuto come obiettivo bug di implementazione degli smart contract". L. H. Newman, *Blockchains have a 'bridge' problem, and hackers know it*, Wired, 2022.

[23]Ronin's Newsletter, *Community Alert: Ronin Validators Compromised*, 2022.

mico, in cui gli utenti usufruiscono di servizi erogati da provider perlopiù centralizzati. La situazione appena descritta si sta ripresentando anche per il Web3: infatti, "anche se le blockchain sono decentralizzate, i servizi che interagiscono con queste sono controllate da un piccolo numero di compagnie private. In effetti, l'industria che emerge a supporto del web decentralizzato è altamente integrata, ciò mina potenzialmente le promesse del Web3."[24]. Alcuni esempi di servizi centralizzati, o scarsamente decentralizzati, che si interfacciano con le blockchain sono: il wallet MetaMask, la nota piattaforma di NFT OpenSea, il provider Infuria e i bridge citati in precedenza. Ciò conferma la tesi di S. Nover e infatti, come lo stesso autore afferma, "poche compagnie dominanti stanno ottenendo un enorme market share costruendo piattaforme user-friendly per portare la nuova iterazione di internet alle masse."[25]. In definitiva, al crescere della semplicità di utilizzo del Web3, il grado di decentralizzazione dei servizi diminuisce.

Quanto descritto porta a concludere che i principi alla base degli attacchi presentati restano validi anche nel caso di bridge che riguardino blockchain non locali. Tuttavia in questi ambiti condurre simili attacchi è più difficile grazie ad una maggiore decentralizzazione; ad esempio sarebbe difficile far leggere a tutti i relayer un evento non realmente avvenuto su una blockchain, come nel caso del ChainId collision attack, in quanto sarebbe necessario ingannare un numero elevato di relayer. Allo stesso modo attacchi come il Rollback Attack richiederebbe la presenza del 51% di nodi malevoli su una blockchain.

In definitiva, se bridge di blockchain, locali e non, soffrono di vulnerabilità simili, nel caso specifico di blockchain locali è più semplice sfruttarle per attaccare il sistema.

[24]S. Nover, *The decentralized web is not decentralized*, Quartz, 2022.
[25]Ibidem.

Capitolo 4

Implementazione e test delle soluzioni

Nel precedente capitolo sono stati discussi alcuni attacchi all'architettura del sistema ed un set di requisiti minimi e consigliati per ottenere un miglior livello di protezione. Appurata la necessità di effettuare alcune modifiche al sistema per renderne più sicuro l'utilizzo, procediamo ora all'implementazione dei requisiti consigliati, elencati nella tabella 3.7, e al testing[1] delle soluzioni implementate.

4.1 Riconfigurazione condizionale del relayer

Come evidenziato nelle sezioni 3.2.1.2, 3.5.1.2 e 3.6.1.2 la sicurezza del sistema può essere migliorata qualora il relayer, ossia il componente che effettua la lettura e trasmissione degli eventi da una blockchain di origine ad una di destinazione, non fosse controllato dall'utente che desidera trasmettere i propri depositi, ma venisse gestito lato blockchain di destinazione, venendo riconfigurato automaticamente solo qualora specifiche condizioni fossero rispettate. Il requisito fondamentale per rimappare il relayer consiste nella presenza di tutti i depositi avvenuti in precedenza all'interno dello storage

[1]Per la realizzazione e la gestione dei test è stata impiegata la libreria `unittest` già inclusa in Python.

del bridge di origine degli eventi; tale condizione è fondamentale per evitare attacchi come il ChainId collision e il Rollback attack.

Rimuovendo all'utente il controllo del relayer è anche possibile aggiungere un ulteriore vincolo, ovvero che vi sia uguaglianza tra il bytecode dei contratti sulla chain di origine del deposito e quella di destinazione; quest'ulteriore verifica consente di identificare modifiche al codice sorgente dei contratti. Individuare tali cambiamenti è di grande importanza in quanto smart contract modificati potrebbero consentire l'invio di eventi di deposito parzialmente veritieri, come ad esempio accade nel caso del FakeLock attack, o di prendere possesso dei fondi degli utenti, come avviene con il CrossCoin Theft Attack.

La richiesta di lettura dei depositi viene effettuata attraverso un'architettura client-server ed ha il seguente flusso di esecuzione:

1. l'utente si reca presso la blockchain di destinazione con un nodo della sua blockchain;

2. viene effettuata una richiesta di remap del relay mediante un client opportunamente configurato per fornire al server che gestisce il relay l'address del bridge di origine e l'id del token di cui si vuole effettuare il trasferimento;

3. il server effettua alcuni controlli sul bytecode degli smart contract e sul contenuto dello storage trie del contratto bridge;

4. qualora il bytecode degli smart contract di origine del deposito fosse differente rispetto a quello dei contratti sulla blockchain di destinazione, o vi fossero depositi mancanti nello storage trie, il relay non verrebbe rimappato; in caso contrario la configurazione del relayer gestito dal server viene modificata, dando inizio alla lettura e trasmissione degli eventi di deposito verso il bridge della chain di destinazione.

Una rappresentazione ad alto livello del processo è visibile nella figura 4.1.

L'implementazione di quanto appena descritto è stata effettuata in Python con l'ausilio della libreria `xmlrpc` per la gestione della comunicazione tra il client ed il server. Alla riga 5 del listato 4.1 è possibile individuare la remote procedure call alla funzione `remap_relayer` da parte del client.

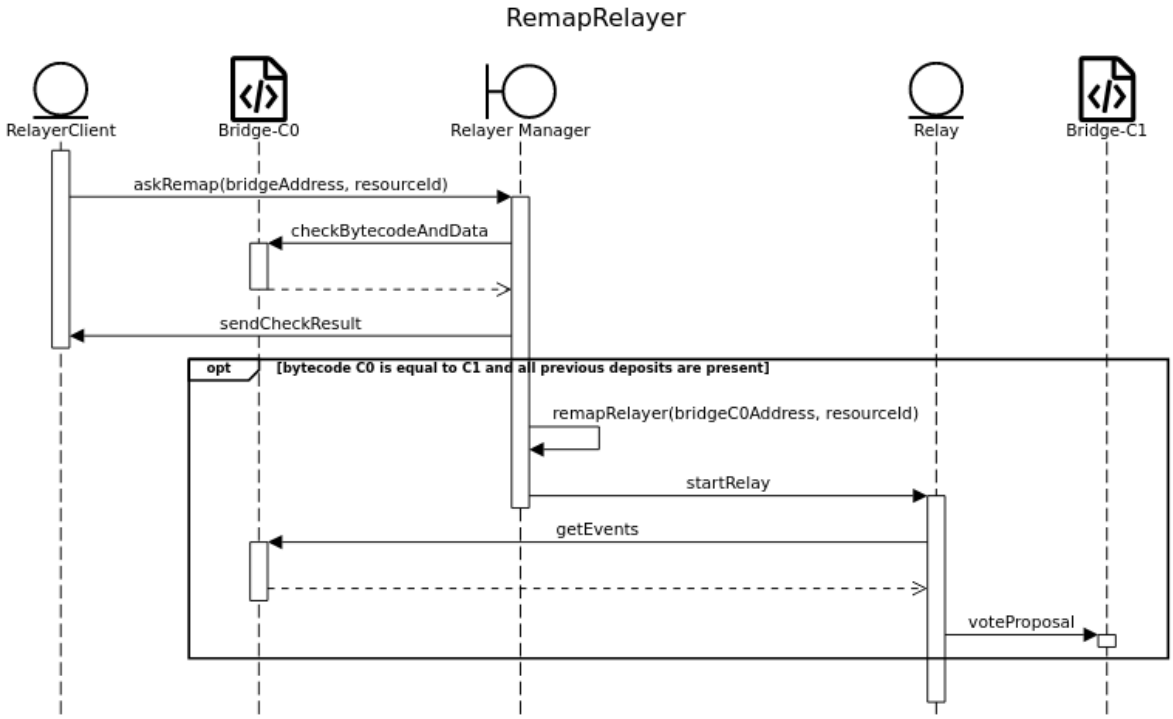


Figura 4.1: Diagramma di sequenza richiesta di remap del relayer.

```
def ask_remap(server_endpoint:str, src_node_endpoint:str, from_block:
    int, bridge_addr:str, res_id:str ):
    with ServerProxy(server_endpoint) as proxy:
        response = proxy.remap_relayer(src_node_endpoint, from_block,
            bridge_addr, res_id)
    return response
```

Listing 4.1: Client per la richiesta di riconfigurazione del relayer da parte dell'utente.

Nel listato 4.2 è invece riportato il contenuto della funzione eseguita dal server una volta che viene ricevuta la chiamata remota. Tra le prime operazioni che vengono svolte si può notare come venga istanziato un Web3 provider per interagire con il nodo dell'utente; in seguito, mediante la funzione `_get_addrs`, vengono recuperati gli indirizzi degli smart contract di cui verrà verificato il bytecode. Tali indirizzi (i.e. `TokenHandler` e `Erc20`) vengono letti direttamente dallo storage del bridge e non sono forniti dall'utente in quanto questo potrebbe indicare address di smart contract con bytecode corretto, ma che non sono quelli per cui è effettivamente configurato il bridge. Ricevendo l'address del bridge, che deve necessariamente essere corretto, pena la non lettura degli eventi da parte del relayer, è possibile richiamare le funzioni `_resourceIDToHandlerAddress` e `_resourceIDToTokenContractAddress`; in questo modo si ha la certezza che verranno restituiti gli indirizzi dei contratti per cui il bridge è realmente configurato.

```
def remap_relayer(self, src_endpoint: str, from_block: int,
    src_bridge_addr: str, res_id: str):
    self.src_endpoint = Web3(HTTPProvider(src_endpoint))
    self._get_addrs(src_bridge_addr, res_id)
    # Check conditions for remapping the relayer
    if self._bytecode_matches() and self._storage_integrity():
        if self.relayer.is_relayer_running():
            self.relayer.stop_relay()
        self.relayer.update_config_json(self.dst_endpoint,
            ContractTypes.ERC20)
        self.relayer.start_relay(latest=from_block)
        return {"response": "OK",
                "extra": "We are parsing your events."}
    else:
        return {"response": "NOK",
```

```
                "extra": "Errors in bytecode verification or in storage
                probing phase."}
```

Listing 4.2: Funzione server side per la riconfigurazione del relayer.

Un'operazione simile viene effettuata anche per il bridge di destinazione e, ottenuti gli indirizzi, si procede recuperando il bytecode dei contratti mediante la chiamata JSON-RPC `eth_getCode`; infine viene verificata l'uguaglianza tra i bytecode, come riportato nel listato 4.3.

```
METADATA_DELIMITER = "264697066735822"
def verify_bytecode(node_endpoints: List[str], src_addrs: List[str],
    dst_addrs: List[str]) -> bool:
    src_bytecode = []
    dst_bytecode = []
    for i in range(0, len(src_addrs)):
        data = json.dumps({"jsonrpc": "2.0", "method": "eth_getCode",
                           "params": [src_addrs[i], "latest"],
                           "id": 1})
        bytecode = json.loads(requests.post(
            url=node_endpoints[0], data=data).text)['result']
        bytecode = bytecode[bytecode.index(METADATA_DELIMITER):]
        src_bytecode.append(bytecode)
        data = json.dumps({"jsonrpc": "2.0", "method": "eth_getCode",
                           "params": [dst_addrs[i], "latest"],
                           "id": 1})
        bytecode = json.loads(requests.post(
            url=node_endpoints[1], data=data).text)['result']
        bytecode = bytecode[bytecode.index(METADATA_DELIMITER):]
        dst_bytecode.append(bytecode)
    if src_bytecode == dst_bytecode:
        logging.info("Bytecode matches")
        return True
    else:
        logging.info("WARNING: Bytecode does not match!")
        return False
```

Listing 4.3: Verifica ugugaglianza bytecode.

È interessante notare come nel listato precedente il bytecode dei contratti non venga confrontato nella sua interezza, ma ne venga estratta solamente una parte, ossia quella relativa ai metadati dello smart contract. I metadati vengono aggiunti dal compilatore di Solidity, che "aggiunge per default l'hash IPFS del file di metadati alla fine del bytecode di ogni contratto"[2]. Nel file di metadati sono presenti numerose informazioni, tra cui l'hash dei contratti; modificare il codice sorgente dello smart contract comporta quindi il cambiamento dei metadati ad esso associati e, di conseguenza, dell'hash IPFS del file. Per questo motivo l'uguaglianza degli smart contract viene verificata utilizzando solamente la parte finale del bytecode estratto dalla blockchain. Questa modalità di verifica può essere ulteriormente estesa recuperando il file di metadati da IPFS e svolgendo su di esso ulteriori controlli.

La verifica dell'integrità dello storage trie fa utilizzo di alcune componenti off-chain ed on-chain. Alla prima richiesta di deposito che si riceve da una blockchain, viene creato un file JSON in cui sono salvati il nonce e i dati relativi al trasferimento; a partire da tali informazioni è calcolato[3] il Merkle trie relativo allo storage del contratto, la root dell'albero viene quindi salvata on-chain su un contratto chiamato `RootBoard`. La `RootBoard` contiene un mapping, riportato nel listato 4.4, che, ad ogni chainId, associa un oggetto `MerkleRoot` e l'update dei valori del mapping è subordinato a una verifica dei privilegi di scrittura, che viene effettuata tramite un modificatore di accesso sulla funzione `setLatestRoot`; in questo modo è possibile verificare che la scrittura dell'ultimo valore non sia libera, ma avvenga solo da parte di utenti autorizzati.

```
...
struct MerkleRoot {
    uint256 latestDepositNonce;
    string trieRoot;
}
//chainId -> latest merkleRoot
mapping(uint256 => MerkleRoot) private chainRoots;
```

[2] *Contract Metadata.* 2017. Solidity Docs. Per ulteriori informazioni sui metadati relativi agli smart contract si rimanda alla documentazione di Solidity `https://docs.soliditylang.org/en/latest/metadata.html`.

[3] Per la creazione e la verifica dei Merkle trie via Python viene utilizzata la libreria `MerkleTools`; per ulteriori informazioni si rimanda a: `https://github.com/Tierion/pymerkletools`.

```
...
```

Listing 4.4: Mapping per il salvataggio on-chain della root del Merkle trie.

A partire dal secondo trasferimento proveniente da una blockchain di origine già conosciuta, viene ricostruito il trie relativo ai depositi leggendo il file JSON dei trasferimenti precedenti e la Merkle root risultante viene poi confrontata con quella salvata on-chain per verificare che le informazioni da cui è stato ricostruito il Merkle trie siano effettivamente corrette; qualora la root del trie appena ripristinato coincida con quella salvata sulla `RootBoard`, viene generata una Merkle proof a partire dai dati presenti sul nodo dell'utente per l'ultimo trasferimento prima di quelli da approvare. Generata la Merkle proof, questa è verificata sul trie del server; qualora la verifica andasse a buon fine ciò significherebbe che lo storage trie del contratto di origine dei nuovi depositi non ha subito alterazioni e dunque le nuove richieste saranno accettate. A questo punto i dati contenuti nel JSON memorizzato off-chain vengono aggiornati per comprendere gli ultimi trasferimenti e la root del nuovo trie è scritta sulla `RootBoard`.

Nel listato 4.5 vengono riportati alcuni dei punti più significativi del procedimento appena descritto.

```
def _storage_integrity(self):
    ...
    # If chain is known we verify the content of the storage
    if os.path.exists(trie_path):
        with open(trie_path) as f:
            saved_trie = json.load(fp=f)
            trie = MerkleTools()
            for deposit in saved_trie["deposits"]:
                trie.add_leaf(deposit["data"], True)
            trie.make_tree()
            chain_root = self._get_root()
            # If the restored trie has a root equal to the one
            # saved on chain we continue
            if trie.get_merkle_root() == chain_root[1]:
                ...
            else:
                return False
            ...
```

```
            # Verify merkle proof on our trie
            user_trie = trie_maker(self.src_endpoint,
                self.dst_endpoint, self.src_addrs[0],
                latest_nonce_saved)
            proof = proof_maker(user_trie, latest_nonce_saved)
            if trie.validate_proof(proof[0], proof[1],
                trie.get_merkle_root()):
                logging.info("Proof is compatible!")
            else:
                logging.info("The proof is not compatible!")
                return False
        ...
        # If this is the first time we see this blockchain or
        # the previous control phase was ok we write the data
        # off-chain and on-chain
        trie = trie_maker(self.src_endpoint, self.dst_endpoint,
            self.src_addrs[0], save_on_disk=True)
        result = self._write_root(saved_trie["deposits"][-1] ["nonce"],
            trie.get_merkle_root())
        ...
```

Listing 4.5: Verifica dello storage trie del bridge di origine del deposito.

4.1.1 Test implementazione

Il remapping condizionale del relayer viene testato prendendo in considerazione tre scenari:

1. il nodo della chain dell'utente che vuole trasmettere i depositi dispone di smart contract il cui bytecode è identico a quello della blockchain di destinazione e nello storage del bridge sono presenti tutti i depositi effettuati in precedenza;

2. il bytecode degli smart contract sul nodo dell'utente differisce rispetto a quello dei contratti sulla chain di destinazione;

3. lo storage del bridge di origine del deposito contiene transazioni differenti rispetto a quelle registrate sullo storico mantenuto dal server che gestisce il relay.

```
test_05askRemapGoodStorageAndCode (__main__.TestPatches) ... ok
test_06askRemapWrongCode (__main__.TestPatches) ... ok
test_07askRemapWrongStorage (__main__.TestPatches) ... ok
```

Figura 4.2: Esito dei test relativi alla riconfigurazione del bridge.

Il funzionamento del server che gestisce la riconfigurazione del relay è stato verificato rispetto ai casi appena descritti mediante una suite di test. Nel caso del primo scenario il test si considera superato se il server rimappa il relayer per leggere gli eventi provenienti dal bridge di origine dell'utente; i test seguenti vengono invece superati se il relayer non viene riconfigurato. Nella figura 4.2 è possibile vedere l'esito dei test.

4.2 Amministrazione distribuita

L'amministrazione distribuita del bridge, descritta nelle sezioni 3.1.1.4 e 3.4.1.1, è stata implementata a partire dal contratto `Governor` di OpenZeppelin[4]. Il funzionamento dell'amministrazione distribuita si basa su un procedimento in quattro step: delega dei privilegi di gestione del bridge alla governance, pubblicazione di una proposta, votazione della proposta ed esecuzione della stessa.

La procedura di deploy del contratto bridge sulla blockchain assegna i privilegi di amministratore all'indirizzo che ha inviato la transazione per la creazione dello smart contract; per rendere l'amministrazione del bridge distribuita è necessario che i privilegi di amministrazione vengano concessi al contratto `BridgeGovernance`, da cui poi verranno votate ed eseguite tutte le funzionalità inerenti al bridge (e.g. aggiunta di relayer, aumento del numero minimo di voti necessari per avviare un deposito, etc.). Il trasferimento dei diritti di amministrazione al contratto `BridgeGovernance` avviene con una chiamata alla funzione `renounceAdmin`, già presente nel contratto bridge, fornendo l'address dello smart contract di amministrazione distribuita. L'operazione appena descritta viene effettuata *una tantum*.

[4]Per ulteriori informazioni sull'implementazione di governance sulla base delle librerie di OpenZeppelin si rimanda a: `https://docs.openzeppelin.com/contracts/4.x/governance`.

La gestione del bridge, a seguito del trasferimento dei privilegi di amministrazione alla governance, avviene mediante un meccanismo basato su proposte e votazioni; per effettuare proposte e votarle è necessario essere parte dell'amministrazione distribuita. Come ipotizzato nella sezione 3.1.1.4, gli admin effettuano una operazione di committment di fondi per garantire il loro corretto comportamento; dunque, per entrare a far parte della governance, è necessario inviare un preciso quantitativo di WEI alla funzione `join`, specificando l'address che si vuole aggiungere alla governance.

Una proposta consiste in una sequenza di azioni, che la governance, a seguito di un periodo di voto, decide se eseguire o meno; la sequenza di azioni viene espressa attraverso attraverso tre liste di parametri: gli indirizzi degli smart contract interessati dalla proposta, un elenco di quantitativi di fondi da inviare insieme alla proposta e una lista di funzioni da chiamare nel momento in cui la proposta viene eseguita[5]. Quanto appena descritto consente ai membri dell'amministrazione distribuita di conoscere e valutare in anticipo gli effetti che l'esecuzione di una proposta avrà sul bridge. L'approccio mediante codifica dei dati che verranno utilizzati per chiamare la funzione di cui si sta votando l'esecuzione consente di utilizzare automaticamente tutte le funzionalità del bridge senza dover scrivere codice aggiuntivo, ad esclusione di quello per codificare e pubblicare una proposta di azione sullo smart contract della governance: il contratto governance diventa quindi un proxy tramite cui interagire con il bridge.

La funzione per pubblicare una proposta sul contratto `BridgeGovernance`, e la codifica della chiamata del metodo e dei parametri necessari per il remap di un bridge, è visibile nel listato 4.6. Per questioni di spazio e di ripetitività non vengono riportati i metodi per la codifica delle altre funzioni di amministrazione del bridge, che differiscono solamente per il parametro `fn_name`.

```
1 def _make_proposal(self, targets: List[str], amount: List[int],
    calldata: List[str], description: str):
2     function = self.bridge_governance.functions.propose(
3         targets, amount, calldata, description)
4     receipt = self._call_function(function)
```

[5]La lista di funzioni che verranno chiamate comprende anche i parametri che verranno passati in input; tale lista viene ottenuta concatenando l'hash della firma della funzione con i parametri della chiamata.

```
    if receipt["status"] == 1:
        event = bridge.bridge_governance.events.ProposalCreated
                .getLogs(fromBlock=receipt["blockNumber"])
        proposal_id = str(event[0]["args"]["proposalId"])
        proposal = {"proposal_id": proposal_id,
                    "targets": targets, "values": amount,
                    "calldata": calldata,
                    "description": description,
                    "description_hash": Web3.keccak(text=description)
                    .hex()}
        with open(os.path.join(PROPOSALS_BASE_PATH, proposal_id
        + ".json"), "w") as f:
            json.dump(obj=proposal, fp=f)
        logging.info("Your proposal id is: " + str(proposal_id))
        return int(proposal_id)

def remap_proposal(self, resource_addresses: List[str], description:
    str):
     calldata = self.bridge.encodeABI(
         fn_name="adminSetResource", args=resource_addresses)
     return self._make_proposal([self.bridge.address], [0], [calldata],
    description)
```

Listing 4.6: Invio di una proposta all'amministrazione distribuita.

La fase di votazione di una proposta presenta alcune peculiarità a seconda di come viene impostata l'amministrazione distribuita. È infatti possibile assegnare un peso al voto di ogni singolo utente sulla base del possesso di un determinato numero di token ERC20 associati alla governance; tuttavia, per lo scenario di amministrazione di un bridge, è stato deciso di assegnare un potere di voto identico a tutti i partecipanti. Impostazioni come il numero di voti disponibili per ogni indirizzo, la durata del periodo di voto, il numero di voti richiesto per raggiungere il quorum, etc. sono specificate mediante alcuni parametri forniti in fase di deploy al costruttore dello smart contract e tramite l'override di specifiche funzioni del contratto `Governance`. Alcuni degli override, giudicati più significativi, sono stati riportati nel listato 4.7.

```
function quorum(uint256 blockNumber) public view override returns
    (uint256) {
```

```
3       return nPartecipants * (quorumPercentage / 100);
4   }
5
6   function _getVotes(address account, uint256 blockNumber, bytes memory
7       params)internal view override returns (uint256) {
8       return 1;
9   }
10
11  function votingDelay() public pure override returns (uint256) {
12      return 0;
13  }
14
15  function votingPeriod() public pure override returns (uint256) {
16      return 10;
17  }
```

Listing 4.7: Configurazione governance.

Come possibile notare dalle righe 12 e 16 del listato precedente, una proposta può essere preceduta da un periodo in cui non può essere votata - ciò risulta utile per dare a una comunità il tempo di discutere una proposta e prepararsi al voto - e da un periodo in cui è possibile votare la proposta; la durata di entrambi i momenti viene specificata attraverso il numero di blocchi che intercorre da quando la proposta viene pubblicata[6].

La votazione, come l'invio di una proposta, è stata protetta mediante l'aggiunta di un modificatore di accesso, denominato `onlyPartecipant`, che consente di far votare e proporre modifiche al bridge solo a coloro che hanno inviato la somma necessaria per poter divenire amministratori, come richiesto dai requisiti consigliati del capitolo precedente.

Una volta terminato il periodo di voto, qualora il quorum fosse stato raggiunto e fossero presenti una maggioranza di voti positivi, la proposta viene eseguita chiamando il metodo `execute` e specificando la proposal che si vuole attuare.

[6]Nel caso in esame, il voting delay è nullo e il numero di blocchi per votare è pari a 10, che equivale a circa 20 secondi. Tempi così brevi aiutano a velocizzare la fase di test, tuttavia in un contesto di utilizzo reale potrebbe essere utile concedere un tempo maggiore ai membri della governance.

```
test_00proposalNonAdmin (__main__.TestPatches) ... ok
test_01proposalAdmin (__main__.TestPatches) ... ok
test_02executeProposalNoQuorum (__main__.TestPatches) ... ok
test_03executeProposalQuorum (__main__.TestPatches) ... ok
test_04executeProposalNotPassed (__main__.TestPatches) ... ok
```

Figura 4.3: Esito dei test relativi alla governance.

4.2.1 Test implementazione

Per verificare il corretto funzionamento dell'amministrazione distribuita sono stati aggiunti alla suite di test alcuni test case riguardanti la possibilità di effettuare proposte, votarle ed eseguirle.

Nello specifico, le condizioni verificate sono le seguenti:

1. un utente non facente parte della governance non deve poter effettuare proposte. Il test si ritiene superato se una transazione contenente una proposal effettuata da un utente che non è membro dell'amministrazione distribuita viene rigettata;

2. un utente parte della governance può effettuare proposte. Il test viene superato se la transazione relativa a una proposta, effettuata da un componente della governance, va a buon fine;

3. una proposta che non ha raggiunto il quorum non può essere eseguita. Questo test viene superato se il tentativo di esecuzione di una proposta che non ha raggiunto il quorum viene rigettato;

4. una proposta che ha raggiunto il quorum e ha ricevuto una maggioranza di voti positivi deve poter essere eseguita. Il test viene giudicato superato se la transazione di esecuzione viene portata a termine con successo.

5. una proposta che abbia raggiunto il quorum, ma abbia ricevuto una maggioranza di voti negativi non deve essere eseguita. Il superamento del test avviene qualora la transazione che richiede di eseguire la proposal non andasse a buon fine.

I risultati dei test appena descritti sono visibili nella figura 4.3.

4.3 Limitazioni ai rimborsi da parte dell'admin

Le restrizioni ipotizzate nella sezione 3.4.1.3 richiedono alcune modifiche alla funzione `deposit`, in quanto l'attuale implementazione di ChainBridge non mantiene una cronologia dei depositi effettuati dagli utenti[7]; tuttavia, a seconda della versione di ChainBridge che viene istanziata sulla blockchain, una storia dei depositi passati potrebbe essere disponibile.

Non salvare sullo smart contract i dati relativi a un deposito e utilizzare per la lettura dello storico dei depositi il sistema di event log della EVM, che per costruzione consente di risparmiare gas, ma i cui dati non sono accessibili dall'interno dello smart contract, può certamente comportare un risparmio cospicuo in termini di gas, specialmente se la blockchain in questione ha un numero di transazioni per secondo elevato.

Un ulteriore aspetto da considerare consiste nell'introduzione dell'amministrazione distribuita mediante l'utilizzo del contratto `BridgeGovernance`, come descritto nella sezione 4.2. Essendo il withdraw una funzionalità la cui chiamata è ristretta al solo admin del bridge, ciò comporta che un rimborso di una determinata somma ad un utente debba necessariamente essere approvato dalla governance. Per questo motivo una possibile soluzione potrebbe essere quella di non reimplementare una struttura dati per memorizzare la storia dei depositi e limitarsi a inviare una proposta di rimborso alla governance, che potrà poi verificare i dati relativi al rimborso mediante gli eventi generati dal bridge.

Nella valutazione della possibilità di non reintrodurre una cronologia dei depositi, per quanto essa possa comportare un risparmio, va tuttavia considerato che lo scenario in analisi prende in considerazione blockchain locali, che in quanto tali avranno un numero di transazioni per secondo e un costo del gas inferiore a quello di una blockchain utilizzata globalmente: il risparmio in termini di gas, garantito dalla rimozione dello storico dei depositi, sarebbe dunque risibile. Per questo motivo, anche considerando che i dati relativi ai depositi passati vengono utilizzati durante la costruzione del Merkle trie per la verifica dello storage del contratto bridge, verrà ripristinato il salvataggio dei dati di

[7] La cronologia dei trasferimenti era mantenuta da ChainBridge fino alla commit `https://github.com/ChainSafe/chainbridge-solidity/commit/8979e9e242add51b97e2eebaa221cf513e5bdde6#diff-d641d27f43a4e1d3dc731dff26fbbaf62e42388ae1455ec70b9521a1e6f27868L39`; in seguito il mapping `_depositRecords` è stato rimosso per ottimizzare l'utilizzo di gas da parte dello smart contract.

deposito on-chain. Inoltre, un'ulteriore motivazione che depone a favore del ripristino della memorizzazione delle informazioni sui depositi consiste nel fatto che i dati contenuti nel mapping possono essere richiamati per verificare condizioni (i.e. ammontare e beneficiario) sui rimborsi approvati dall'amministrazione distribuita; ciò può essere utile per verificare che un rimborso approvato dalla governance non presenti errori passati inosservati dai votanti.

La cronologia dei depositi viene memorizzata in due mapping innestati: la prima chiave è rappresentata dal nonce del deposito, la seconda dal chainId della blockchain di destinazione del deposito. I valori memorizzati consistono nel quantitativo di token trasferiti e nell'address che ha effettuato l'operazione: tali dati vengono salvati immediatamente a seguito del deposito, come visibile a riga 9 del listato 4.8.

```
1  // nonce -> chainId -> data
2  mapping(uint64 => mapping(uint8 => bytes)) public _depositRecords;
3
4  function deposit(uint8 destinationDomainID, bytes32 resourceID,
5      bytes calldata data, bytes calldata feeData) external payable
6      whenNotPaused {
7      _deposit(destinationDomainID, resourceID, data, feeData);
8      // Save deposit data
9      uint64 depositNonce = _depositCounts[destinationDomainID];
10     _depositRecords[depositNonce][destinationDomainID] = data;
11 }
```

Listing 4.8: Salvataggio informazioni deposito.

Per effettuare una proposta di rimborso è necessario fornire alcune informazioni relative al deposito a cui si sta facendo riferimento, nello specifico sono richiesti: il chainId della blockchain verso cui sono stati inviati i fondi, il nonce del deposito e i dati, in forma di concatenazione di byte, per effettuare il rimborso.

La funzione `adminWithdraw` originale è quindi stata resa richiamabile solo dall'interno dello smart contract attraverso il modificatore `internal`; in questo modo non è possibile accedere alla vecchia funzione che non effettua controlli sui dati passati in input, e, essendo necessari dati aggiuntivi per svolgere le verifiche preliminari sul rimborso, la funzione è stata rinominata, in quanto i parametri aggiuntivi modificano la firma della funzione e non permettono di effettuare un semplice override. Come visibile alle righe

17 e 19 del listato 4.9, i dati per lo svolgimento del rimborso vengono confrontati con quelli memorizzati nella cronologia dei depositi; qualora le informazioni fornite non trovassero riscontro nello storico dei depositi le clausole require produrrebbero un errore, non permettendo di fatto di continuare la procedura di rimborso.

```
function adminWithdraw(uint8 chainId, uint64 depositNonce, address
    handlerAddress, bytes memory data) external onlyAdmin {
    // Data sent by admin
    address token;
    address recipient;
    uint256 amount;
    (token, recipient, amount) = abi.decode(data, (address, address,
        uint256));
    // History
    bytes memory depositData = _depositRecords[depositNonce][chainId];
    uint256 depositAmount;
    uint256 addressLength;
    address sender;
    (amount, addressLength, sender) = abi.decode(depositData, (uint256,
        uint256, address));
    // Checks if the given data matches with the one saved for the
   deposit
    require(recipient == sender, "The refund must be sent to the
        original sender.");
    require(amount == depositAmount,"The refund amount has to be the
        same as the deposited one.");
    _adminWithdraw(handlerAddress, data);
}
```

Listing 4.9: Verifica del deposito e rimborso all'utente.

Come accennato in precedenza, è interessante notare come la presenza del modificatore di accesso `onlyAdmin` comporti che l'azione di rimborso debba essere chiamata necessariamente dall'amministratore del bridge e di conseguenza, visto quanto discusso nella sezione 4.2, votata dalla governance. Ciò aggiunge un'ulteriore livello di protezione dalla discrezionalità del bridge admin, che non può quindi decidere di rimborsare arbitrariamente un utente.

```
test_08refundWrongData (__main__.TestPatches) ... ok
test_09refundGoodData (__main__.TestPatches) ... ok
```

Figura 4.4: Esito dei test relativi all'emissione di rimborsi.

4.3.1 Test implementazione

I test relativi all'emissione dei rimborsi da parte dell'amministrazione distribuita verificano il caso in cui i dati forniti riguardo alla transazione da rimborsare non trovino riscontro nello storico delle transazioni mantenuto dal bridge e il caso in cui i dati relativi al rimborso siano corretti. Nel primo caso il test si considera superato se non viene emesso nessun rimborso, nel secondo se il rimborso viene emesso con successo.

Nella figura 4.4 è visibile l'esito dei test.

4.4 Retribuzione bridge admin periodica

La gestione delle fee per l'utilizzo di un bridge è già integrata all'interno di ChainBridge attraverso lo smart contract `BasicFeeHandler`; tale contratto conserva nel proprio bilancio tutte le fee inviate ad ogni deposito e permette al bridge admin di prelevare da esso i fondi accumulati tramite la funzione `transferFee`. Il metodo appena descritto prende in input due liste contenenti gli indirizzi e la quantità di fondi da inviare ad ogni address.

Come teorizzato nelle sezioni 3.1.1.3 e 3.1.1.4, una retribuzione degli admin periodica può aiutare a garantire un loro comportamento positivo; è quindi possibile bloccare il prelievo delle fee attraverso il calcolo del numero di blocchi intercorsi dall'ultima volta che la funzione è stata richiamata. Nel listato 4.10 viene riportato quanto appena descritto, nello specifico la possibilità di prelievo viene limitata a una volta ogni sei mesi.

```
1 contract FeeHandlerLockTime is BasicFeeHandler {
2     uint256 private latestWithdraw;
3     // 6 months at 30 blocks per minute
4     uint256 constant COOLDOWN = 7776000;
5
6     constructor(address bridgeAddress) BasicFeeHandler(bridgeAddress) {
7         latestWithdraw = block.number;
```

```
test_10withdrawFee (__main__.TestPatches) ... ok
test_11withdrawFeeTooEarly (__main__.TestPatches) ... ok
```

Figura 4.5: Esito dei test relativi al prelievo delle fee.

```
8       }
9
10      function transferFee(address payable[] calldata addrs, uint256[]
11          calldata amounts) external onlyAdmin {
12          require(block.number >= latestWithdraw + COOLDOWN, "You need
13              to wait before you can withdraw.");
14          latestWithdraw = block.number;
15          _transferFee(addrs, amounts);
16      }
17  }
```

Listing 4.10: Bloocco del withdraw per l'admin sulla base di un periodo di tempo predeterminato.

4.4.1 Test implementazione

I test relativi al prelievo delle fee verificano che non sia possibile prelevare denaro prima di un determinato numero di blocchi. Per velocizzare la fase di test è stato impostato il divieto di prelevare fee prima che siano passati 10 blocchi dall'ultimo prelievo: il primo test deve quindi passare, in quanto in precedenza non sono stati prelevati fondi, al contrario il secondo test viene giudicato positivo se il prelievo viene negato, considerato che il withdraw precedente è avvenuto da un numero di blocchi inferiore a quello impostato sullo smart contract[8].

L'esito dei test appena descritti è riportato nella figura 4.5.

[8]Il quantitativo di blocchi che intercorre tra il primo test e il secondo dipende dalla velocità di esecuzione dei test e dal block production rate della blockchain; nel caso in questione, perchè trascorrano i 10 blocchi necessari a poter effettuare una nuova richiesta di prelievo delle fee, è richiesto più tempo di quello che intercorre tra l'esecuzione del primo e del secondo test.

Conclusioni

Nella presente tesi è stato proposto l'utilizzo di blockchain locali per portare mezzi di pagamento digitali e tecnologie innovative in zone del mondo ove una connessione a internet non fosse presente, risultasse troppo costosa per gli abitanti della zona o, essendo inaffidabile, non permettesse l'utilizzo di chain già esistenti.

L'utilità delle blockchain locali risulterebbe però limitata se queste rimanessero totalmente isolate dal resto del mondo; pertanto, per connettere tali sistemi e consentire agli utenti di spostare i loro token su qualsiasi chain, a prescindere che questa sia locale o meno, è stato proposto l'utilizzo di tecnologie che consentono l'interoperabilità tra blockchain.

In seguito, è stato istanziato un prototipo del sistema descritto facendo uso di due tecnologie già esistenti, ovvero: Polygon-Edge SDK, per la creazione delle chain locali, e ChainBridge, per interfacciare le blockchain. Questo ha permesso di appurare come le particolari condizioni delle blockchain locali implichino alcune problematiche derivanti dalla bassa decentralizzazione di questi sistemi nonchè dal possibile opportunismo degli agenti che vi operano e come ciò possa rappresentare un pericolo per il sistema nel suo complesso. A partire da questi presupposti sono stati testati con successo sei possibili attacchi, suddivisi in tre macro-categorie (i.e. fund stealing, double spending and bridge poisoning), nei confronti del sistema. Da una successiva analisi è poi emerso come questi attacchi possano essere estesi anche ad altri bridge tra blockchain; tuttavia le particolari condizioni di utilizzo delle chain locali tendono a renderne più semplice l'esecuzione in tali contesti.

A partire dagli attacchi presentati, sono stati stilati alcuni requisiti con l'obiettivo di mitigare le vulnerabilità individuate. I principali requisiti individuati sono:

- la presenza di un sistema di amministrazione distribuita per impedire che un solo individuo possa modificare arbitrariamente le impostazioni del bridge;
- l'aggiunta di incentivi e disincentivi economici per gli amministratori dei bridge;
- l'adozione di un approccio proattivo nel valutare se un relay debba essere rimappato o meno per la lettura degli eventi di deposito provenienti da una blockchain locale;
- l'aggiunta di ulteriori controlli per ridurre i privilegi degli amministratori dei bridge.

L'implementazione dei requisiti è stata svolta apportando alcune modifiche ai contratti di ChainBridge e aggiungendo nuove componenti al sistema, come, ad esempio, il relay manager e un sistema di governance distribuita dei bridge. Il corretto funzionamento dell'implementazione dei requisiti è stato poi verificato attraverso una test suite, che non ha evidenziato criticità nell'implementazione.

Per quanto il sistema sia stato testato in uno scenario con caratteristiche simili a quelle del suo ipotetico utilizzo reale, tale scenario rimane tuttavia artificiale e ciò potrebbe non aver evidenziato alcune criticità nell'architettura del sistema, che possono portare ad ulteriori attacchi; pertanto un periodo di test in un contesto d'uso reale potrebbe essere auspicabile per individuare vulnerabilità aggiuntive.

Inoltre, la verifica del bytecode dei contratti ad opera del relay manager, pur consentendo di individuare smart contract con codice sorgente modificato, non permette di interfacciare bridge che usino versioni di ChainBridge più nuove o che abbiano subito patch da parte del team di sviluppo. Sotto questo punto di vista ciò cristallizza il sistema e non permette di introdurre miglioramenti futuri agli smart contract. Una possibile soluzione è quella di introdurre un sistema di controllo più articolato da parte del relayer manager, ad esempio, verificando che l'indirizzo che ha effettuato il deploy del bridge rientri all'interno di una lista di quelli appartenenti all'autorità che gestisce i bridge, oppure utilizzando una suite di test, che, una volta ottenuto il bytecode degli smart contract, verifichi il corretto funzionamento del bridge di origine del deposito a prescindere dalla sua versione.

Un ulteriore aspetto che meriterebbe di essere approfondito consiste nello studio di incentivi per la condivisione dello stato dei bridge, a prescindere che si attenda di ricevere

depositi dalle blockchain che questi servono. Ciò consentirebbe al relay manager di una determinata chain locale di disporre anticipatamente delle informazioni da cui costruire la Merkle proof per verificare che i depositi passati siano ancora presenti nella catena con cui si sta per interagire per la prima volta, migliorando ulteriormente la sicurezza del sistema nel suo complesso.

Ringraziamenti

Nei due anni di questo percorso di Laurea Magistrale ho avuto modo di mettermi alla prova e affrontare nuove sfide. Tutto ciò non sarebbe stato possibile senza il supporto di alcune persone, che meritano di diritto un piccolo spazio in questa tesi.

Un ringraziamento va al Prof. Ferretti e al Dott. Zichichi per avermi seguito nella scrittura della tesi, i loro consigli sono stati di fondamentale importanza per dare una direzione a questo lavoro.

Grazie a Michele ed Emanuele per aver collaborato con me ad alcuni dei progetti più impegnativi di questi ultimi anni. La vostra pazienza nei confronti della mia pignoleria è stata encomiabile.

Grazie ad Elia e Gianluigi per aver inconsapevolmente fornito l'hardware su cui sono state fatte girare le blockchain locali e su cui è stato svolto lo sviluppo e il test delle soluzioni.

Ritornando in ambito "crypto", grazie a Shine e ad Asfodel per aver partecipato al progetto più pixellato di sempre, che mi ha permesso di muovere i primi passi in ambito blockchain.

Grazie a tutto il team di sviluppo di ChainBridge e del PolygonEdge-SDK per aver risposto alle mie domande e aver risolto alcuni dubbi sul funzionamento degli strumenti utilizzati in questa tesi.

Un ringraziamento speciale va alla mia famiglia, che in questi anni mi ha supportato nei momenti di maggiore fatica e stanchezza.

Infine, l'ultimo ringraziamento va a Benedetta, l'unica capace di rischiarire anche le giornate più grige.

Bibliografia

[1] E. Barinaga. «Sarafu: A cryptocurrency for kenyan rural communities». In: *SAGE Business Cases* (2022). A cura di SAGE Publications. URL: https://sk.sagepub.com/cases/sarafu-a-cryptocurrency-for-kenyan-rural-communities.

[2] Crypto Baristas. *Holder Perks*. 2022. URL: https://cryptobaristas.com/holder-perks/.

[3] R. Belchior et al. «A Survey on Blockchain Interoperability: Past, Present, and Future Trends». In: *CoRR* (2021). URL: https://arxiv.org/abs/2005.14282.

[4] R. Belchior et al. «Do You Need a Distributed Ledger Technology Interoperability Solution?» In: *TechRxiv* (2022). URL: https://www.techrxiv.org/articles/preprint/Do_You_Need_a_Distributed_Ledger_Technology_Interoperability_Solution_/18786527.

[5] M. Binder. «Bored Ape Yacht Club caused Ethereum fees to soar to astronomical levels». In: *Mashable* (2022). URL: https://mashable.com/article/ethereum-gas-fees-skyrocket-bored-ape-yacht-club-otherside-nft-launch.

[6] V. Buterin. *[AMA] We are the EF's Research Team (Pt. 7: 07 January, 2022)*. 2022. URL: https://old.reddit.com/r/ethereum/comments/rwojtk/ama_we_are_the_efs_research_team_pt_7_07_january/hrngyk8/.

[7] V. Buterin. «Chain Interoperability». In: *R3 Reports* (2016). URL: https://www.r3.com/wp-content/uploads/2017/06/chain_interoperability_r3.pdf.

[8] V. Buterin. *Crypto Cities*. 2021. URL: https://vitalik.ca/general/2021/10/31/cities.html.

[9] V. Buterin. *Merkling in Ethereum*. A cura di Ethereum fundation blog. 2015. URL: https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/.

[10] V. Buterin. *The Limits to Blockchain Scalability*. 2021. URL: https://vitalik.ca/general/2021/05/23/scaling.html.

[11] V. Buterin e F. Vogelsteller. *EIP-20 Token Standard*. 2015. URL: https://eips.ethereum.org/EIPS/eip-20.

[12] CertiK. *Wormhole Bridge Exploit Analysis*. 2022. URL: https://certik.medium.com/wormhole-bridge-exploit-analysis-5068d79cbb71.

[13] ChainSafe. *ChainBridge*. 2020. URL: https://chainbridge.chainsafe.io/.

[14] ChainSafe. *chainbridge-solidity*. 2020. URL: https://github.com/ChainSafe/chainbridge-solidity/tree/master/contracts.

[15] Blockstream corporation. *Blockstream*. 2022. URL: https://blockstream.com/.

[16] IPFS Docs. *What is IPFS*. 2019. URL: https://docs.ipfs.io/concepts/what-is-ipfs/#decentralization.

[17] Solidity Docs. *Contract Metadata*. 2017. URL: https://docs.soliditylang.org/en/latest/metadata.html.

[18] W. Entriken et al. *EIP-721 Non-Fungible Token Standard*. 2018. URL: https://eips.ethereum.org/EIPS/eip-721.

[19] C. Faife. «Explaining Crypto's Billion-Dollar Bridge Problem». In: *The Verge* (2022). URL: https://www.theverge.com/23017107/crypto-billion-dollar-bridge-hack-decentralized-finance.

[20] S. Ferretti et al. «Incentivized Data Mules Based on State-Channels». In: *IEEE* (2022). URL: https://cris.unibo.it/handle/11585/882405.

[21] P. De Filippi, C. Wray e G. Sileno. «Smart Contracts». In: *Internet Policy Review* 10.2 (2021). URL: https://policyreview.info/glossary/smart-contracts.

[22] P. Frauenthaler et al. *Leveraging Blockchain Relays for Cross-Chain Token Transfers*. 2020. URL: https://www.researchgate.net/publication/339400544_Leveraging_Blockchain_Relays_for_Cross-Chain_Token_Transfers.

[23] K. Jezek. *Ethereum Data Strucutres*. 2020. URL: https://arxiv.org/pdf/2108.05513.pdf.

[24] S. Johnson, P. Robinson e J. Brainard. «Sidechains and interoperability». In: *CoRR* (2019). URL: https://arxiv.org/abs/1903.04077.

[25] O. Kharif. «Hackers Steal About $600 Million in One of the Biggest Crypto Heists». In: *Bloomberg* (2022). URL: https://www.bloomberg.com/news/articles/2022-03-29/hackers-steal-590-million-from-ronin-in-latest-bridge-attack.

[26] libp2p. *Addressing*. 2021. URL: https://docs.libp2p.io/concepts/addressing/.

[27] Y. Lin. *Istanbul Bizantine Fault Tolerance*. 2017. URL: https://github.com/ethereum/EIPs/issues/650.

[28] Y. Lin et al. *BcMON: Blockchain Middleware for Offline Networks*. A cura di arXiv. 2022. URL: https://arxiv.org/abs/2204.01964.

[29] D. Maldonado-Ruiz et al. *Secure and Internet-Less Connectivity to a Blockchain Network for Limited Connectivity Bank Users*. 2020. URL: https://www.researchgate.net/publication/344494480_Secure_and_Internet-Less_Connectivity_to_a_Blockchain_Network_forLimited_Connectivity_Bank_Users.

[30] R. Mioli. *bc_connector*. 2022. URL: https://github.com/NorwegianGoat/bc_connector.

[31] R. Mioli. *edge_utils*. 2022. URL: https://github.com/NorwegianGoat/edge_utils.

[32] multiformats. *Self-describing network addresses*. 2017. URL: https://github.com/multiformats/website/blob/master/content/multiaddr.md.

[33] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2009. URL: https://bitcoin.org/bitcoin.pdf.

[34] Ronin Network. *Community Alert: Ronin Validators Compromised*. 2022. URL: https://roninblockchain.substack.com/p/community-alert-ronin-validators?s=w.

[35] L. H. Newman. «Blockchains have a 'bridge' problem, and hackers know it». In: *Wired* (2022). URL: https://wired.me/business/cryptocurrency/blockchains-have-a-bridge-problem-and-hackers-know-it/.

[36] S. Nover. «The decentralized web is not decentralized». In: *Quartz* (2022). URL: https://qz.com/2112965/web3-is-not-decentralized/.

[37] OpenZeppelin. *How to set up on-chain governance.* 2021. URL: https://docs.openzeppelin.com/contracts/4.x/governance.

[38] S. Pini. «Alfa Romeo Tonale, ecco perché NFT aumentano il valore residuo». In: *Il sole 24 ore* (2022). URL: https://www.ilsole24ore.com/art/alfa-romeo-tonale-ecco-perche-nft-aumentano-valore-residuo-AEXPjBEB.

[39] P. Robinson. «Survey of Crosschain Communications Protocols». In: *CoRR* (2021). URL: https://arxiv.org/abs/2004.09494.

[40] K. Sigalos. «More than $320 million stolen in latest apparent crypto hack». In: *CNBC* (2022). URL: https://www.cnbc.com/2022/02/02/320-million-stolen-from-wormhole-bridge-linking-solana-and-ethereum.html.

[41] R. Stevens. *What Are Wrapped Tokens?* A cura di CoinDesk. 2022. URL: https://www.coindesk.com/learn/what-are-wrapped-tokens/.

[42] Tierion. *pymerkletools*. 2016. URL: https://github.com/Tierion/pymerkletools.

[43] Trapesys. *Polygon Edge*. 2021. URL: https://edge-docs.polygon.technology/docs/overview.

[44] International Telecommunication Union. *Measuring digital development Facts and figures*. A cura di ITUPublications. 2021. URL: https://www.itu.int/en/ITU-D/Statistics/Documents/facts/FactsFigures2021.pdf.

[45] usability.gov. *Personas*. 2022. URL: https://www.usability.gov/how-to-and-tools/methods/personas.html.

[46] M. Westerkamp e M. Diez. «Verilay: A Verifiable Proof of Stake Chain Relay». In: *CoRR* (2022). URL: https://arxiv.org/abs/2201.08697.

[47] Bitcoin Wiki. *Scalability*. 2020. URL: https://en.bitcoin.it/wiki/Scalability.

[48] Ethereum Wiki. *Patricia Trie*. 2020. URL: https://eth.wiki/en/fundamentals/patricia-tree.

[49] Ubutu Wiki. *Uncomplicated Firewall*. 2007. URL: https://wiki.ubuntu.com/UncomplicatedFirewall.

[50] Wikipedia. *Blockchain*. 2022. URL: https://en.wikipedia.org/wiki/Blockchain.

[51] J. Xu, D. Ackerer e A. Dubovitskaya. *A Game-Theoretic Analysis of Cross-Chain Atomic Swaps with HTLCs*. 2021. URL: https://www.researchgate.net/publication/346143775_A_Game-Theoretic_Analysis_of_Cross-Chain_Atomic_Swaps_with_HTLCs.

[52] M. Zichichi, B. Distefano e N. Pocher. «MOATcoin: exploring challenges and legal implications of smart contracts through a gamelike DApp experiment». In: *ACM* (2020). URL: https://dl.acm.org/doi/abs/10.1145/3410699.3413798.