UNIVERSITÀ DEGLI STUDI DI BERGAMO

SCUOLA DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica

Classe n. 21-270

Un approccio alla Timed-Release Encryption basato su Smart Contract

Relatore: Chiar.mo Prof. Stefano Paraboschi

Prova finale di Andrea Cattaneo

Matricola n. 1040655

ANNO ACCADEMICO

2017/2018

If I have 300 ideas in a year and
just one turns out to work I am satisfied.
Alfred Nobel

Sommario

Questa tesi si pone come obiettivo la risoluzione del problema dell'invio di un messaggio "nel futuro". La soluzione proposta utilizza la blockchain come orologio autoritativo e sfrutta alcuni smart contract per orchestrare le parti coinvolte.

Nella parte introduttiva presenteremo le reti blockchain. Partiremo dalla storia della blockchain, elencheremo le principali tecniche per il raggiungimento del consenso e la classificazione in blockchain *permissioned/permissionless*. Successivamente analizzeremo alcuni dati sul livello di adozione della tecnologia blockchain nelle organizzazioni. Passeremo poi agli Smart Contract, definendoli e mostrando alcuni accorgimenti da seguire per la loro creazione. Andremo poi a discutere la Timed-Release Encryption. Successivamente presenteremo il protocollo proposto per ottenere la Timed-Release Encryption. Inoltre ne realizzeremo una *proof of concept* sullo Stellar Network. A seguire proporremo un modello per un analisi quantitativa della robustezza delle implementazioni del protocollo. Nell'ultima parte vedremo alcuni possibili usi della Timed-Release Encryption.

Indice

Elenco delle figure

Capitolo 1

Introduzione

La Blockchain è uno degli argomenti più caldi nell'industria dell'IT. In questo capitolo cercheremo di fare chiarezza, partendo dalle origini di questa tecnologia sino ad arrivare allo stato dell'arte odierno. Il contenuto delle sezioni seguenti è basato sull'articolo *Blockchain technology overview* del *NIST* [47].

1.1 Cosa sono le blockchain

Le blockchain sono registri digitali *tamper evident*[1] e *tamper resistant*[2] implementati in un ambiente distribuito. Solitamente non vi è la presenza di autorità centrali come banche, società o governi. Al loro livello base consentono a una community di utenti di registrare transazioni in un *ledger* condiviso. Inoltre le blockchain (in condizioni di normale funzionamento) garantiscono che nessuna transazione possa essere manomessa dopo essere stata pubblicata.

[1] che rendono evidenti eventuali manomissioni

[2] resistenti a manomissioni

Nel 2008 l'idea di blockchain è stata combinata con altre tecnologie e concetti informatici per creare le criptovalute: denaro elettronico protetto e garantito attraverso meccanismi crittografici anziché tramite autorità centrali.

1.2 Storia della blockchain

I concetti chiave alla base della tecnologia blockchain sono emersi a cavallo tra gli anni '80 e '90.

Nel 1989 Lamport sviluppò il protocollo Paxos. Si tratta di un protocollo per il raggiungimento di un consenso in una rete di computer dove i computer o la rete stessa posso essere inaffidabili. Lo descrisse nel paper *The Part-Time Parliament* [23], che venne pubblicato solo nel 1998.

Nel 1991 Haber e Stornetta proposero l'uso di una catena di hash per risolvere il problema del *time-stamping* di documenti, ossia per poter permettere di dimostrare l'esistenza di un certo documento in un istante di tempo passato [17].

Questi concetti sono stati combinati e applicati al denaro elettronico e descritti nel paper *Bitcoin: A peer-to-peer electronic cash system* [30], pubblicato nel 2008 sotto lo pseudonimo Satoshi Nakamoto. Sulla base di questo documento nacque nel 2009 il *Bitcoin cryptocurrency blockchain network*. Il paper di Nakamoto contiene lo schema base seguito dalla maggior parte dei sistemi di criptovaluta moderni.

Diversi sistemi di pagamento elettronico esistevano già prima della nascita di Bitcoin (ad esempio ecash e NetCash), ma nessuno di questi raggiunse una vasta adozione. La grande innovazione di Bitcoin rispetto ai precessori è stata la blockchain; questa ha consentito un'implementazione distribuita in grado di evitare un *single point of failure* e di garantire che nessun utente singolo sia in grado di controllare denaro elettronico.

In Bitcoin gli utenti sono anonimi, ma l'identificativo del loro account non lo è. Questa caratteristica prende il nome di *pseudo-anonimità*. A causa della pseudo-anonimità è essenziale disporre di meccanismi in grado di creare fiducia in un ambiente dove gli utenti non possono facilmente essere identificati. Prima della blockchain, questa fiducia veniva tipicamente fornita tramite intermediari e terze parti fidate. La rete blockchain invece garantisce il bisogno di fiducia senza terze-parti fidate sfruttando le seguenti caratteristiche:

- **Ledger**: la blockchain usa un ledger sequenziale per fornire un elenco completo delle transazioni avvenute. A differenza dei database tradizionali, gli elementi delle blockchain non possono essere sovrascritti.

- **Sicurezza**: le blockchain sfruttano la crittografia per garantire che i dati contenuti nel ledger non siano stati alterati e l'attendibilità delle transazioni.

- **Condivisione**: il ledger è condiviso tra diversi partecipanti. Questo garantisce trasparenza tra i nodi della blockchain.

- **Distruzione**: le blockchain possono essere distribuite. Ciò permette di scalare il numero di nodi della rete blockchain in modo da renderla più resistente agli attacchi. In generale, all'aumentare del numero di nodi la capacità di un attaccante di destabilizzare il protocollo di consenso diminuisce.

1.3 Categorie di blockchain

Effettuando una classificazione basata sul modello di permessi, è possibile individuare due categorie di blockchain.

1.3.1 Permissionless

Le reti blockchain *permissionless* sono ledger decentralizzati che danno la possibilità a chiunque di scrivere blocchi, senza il bisogno del permesso esplicito di una autorità. Spesso le piattaforme blockchain permissionless sono software open source, liberamente disponibili a chiunque per poter essere scaricati. Una conseguenza del fatto che chiunque ha la possibilità di aggiungere blocchi è che la blockchain può essere letta da qualunque soggetto, così come chiunque può effettuare transazioni sulla blockchain.

Qualsiasi utente di una blockchain permissionless può leggere e scrivere sul ledger. Poiché chiunque può partecipare alle reti blockchain permissionless, utenti malintenzionati potrebbero tentare di aggiungere blocchi malevoli al fine di sovvertire il sistema. Per prevenire ciò solitamente le blockchain usano algoritmi di consenso basati sul consumo o sul possesso di risorse (si veda la sezione 1.4) . Solitamente i meccanismi di consenso nelle blockchain permissionless promuovono i comportamenti non-malevoli attraverso ricompense in criptovaluta attribuite a chi pubblica blocchi conformi al protocollo.

1.3.2 Permissioned

Nelle blockchain permissioned gli utenti che pubblicano blocchi devono essere autorizzati da una qualche autorità (centralizzata o decentralizzata). Poiché solo gli utenti autorizzati possono operare sulla blockchain, è possibile atturare delle restrizioni sui permessi di lettura e scrittura. Le blockchain permissioned possono essere concepite in modo da permettere a chiunque di leggere la blockchain o possono fornire l'accesso in lettura solo ad alcuni utenti. Possono anche consentire a chiunque di effettuare transazioni sulla blockchain o, ancora, dare questa possibilità solo ad alcuni utenti. Le blockchain permissioned possono essere istanziate e manutenute

attraverso software open source o closed source.

Anche le blockchain permissioned possono avere la stessa tracciabilità degli asset scambiati come nel caso delle blockchain permissionless, così come la stessa struttura distribuita, resiliente e ridondante. Usano inoltre un meccanismo di consenso per la pubblicazione dei blocchi, ma spesso in questi casi non richiedono il consumo o il possesso di risorse (a differenza delle blockchain permissionless). Questo perché l'identità dei partecipanti è nota; gli attori che operano sulla blockchain hanno un certo grado di fiducia l'uno con l'altro e quindi i meccanismi di consenso possono essere meno dispendiosi computazionalmente e più veloci.

Alcune reti blockchain permissioned supportano la capacità di rendere note le informazioni contenute in alcune transazioni solo a specifici utenti. Con questa funzionalità si può ottenere un certo livello di privacy nelle transazioni. Per esempio, sulla blockchain può rimanere una traccia pubblica del fatto che una certa transazione è avvenuta tra due utenti, e allo stesso tempo far si che il contenuto della transazione sia accessibile solo dalle parti coinvolte. Alcune blockchain permissionless richiedono che tutti gli utenti autorizzino l'invio e il ricevimento di transazioni. In questi sistemi le parti lavorano insieme per raggiungere un obiettivo di business condiviso con disincentivi naturali nel commettere frodi (visto che possono poi essere identificati). Se vengono effettuati comportamenti scorretti è possibile individuare le organizzazioni coinvolte e quali strumenti legali sono a disposizione per punire i colpevoli.

1.4 Modelli di consenso

Una feature fondamentale per la tecnologia blockchain è che non vi è bisogno di una terza parte fidata per conoscere lo stato del sistema, infatti ogni utente può verificare l'integrità della blockchain. Per aggiungere un nuovo blocco tutti i nodi devono

raggiungere un consenso condiviso, anche se un disaccordo temporaneo è permesso. Per raggiungere questo obiettivo è necessario implementare un meccanismo di consenso. Nelle blockchain permissionless ci sono in genere diversi nodi che competono per pubblicare il prossimo blocco. Solitamente lo fanno per ottenere ricompense e/o commissioni sulle transazioni. Ogni nodo è motivato dall'ottenere un guadagno economico; non è interessato agli obiettivi degli altri nodi. In questa situazione, perché un nodo dovrebbe propagare un blocco che un altro nodo sta cercando di pubblicare? La blockchain risolve questo problema attraverso un meccanismo di consenso che permette ad un gruppo di utenti reciprocamente diffidenti di lavorare insieme.

1.4.1 Proof of Work (PoW)

Nel modello *proof of work* l'utente che pubblica un blocco è il primo che risolve un puzzle computazionalmente oneroso. La soluzione di questo puzzle è la "prova" del lavoro effettuato. Il puzzle è progettato in modo tale sia difficile trovare una soluzione, ma semplice verificare la validità della soluzione. Questo permette a tutti gli altri nodi di verificare facilmente la validità dei blocchi proposti. Un tipico puzzle utilizzato è cercare un *nonce* da scrivere nell'header del blocco tale per cui l'hash digest del blocco sia minore di un certo valore target. Il valore target può essere modificato nel tempo per modificare la difficoltà del problema al fine di influenzare la velocità di pubblicazione di nuovi blocchi.

Ad esempio Bitcoin, che usa la proof of work, regola la difficoltà del puzzle ogni 2016 blocchi con l'obiettivo di ottenere una velocità pari a circa un blocco ogni 10 minuti. La regolazione avviene essenzialmente aumentando o diminuendo il numero di zeri necessari come prefisso dell'hash digest. All'aumentare del numero di zeri richiesti la difficoltà aumenta, poiché diminuiscono le possibili soluzioni. Viceversa, diminuendo il numero di zeri la difficoltà si riduce, perché lo spazio di soluzioni

diventa più grande. Questa regolazione ha lo scopo di mantenere relativamente costante la difficoltà del puzzle al variare della potenza di calcolo a disposizione, e quindi garantire la sicurezza dell'intera rete Bitcoin.

La proof of work è una tecnica computazionalmente onerosa per progettazione. Una conseguenza è che richiede l'uso di grandi quantità di energia elettrica. Per questo motivo i nodi tendono ad esser posizionati in area geografiche dove l'energia elettrica è economica.

Un importante aspetto di questo modello è che i puzzle devono essere indipendenti tra di loro; in altre parole, il lavoro compiuto per risolvere un puzzle non deve in alcun modo influenzare la facilità di risoluzione dei puzzle successivi. Ciò comporta che quando un utente riceve un blocco completo e valido da un altro utente è incentivato a scartare immediatamente il puzzle a cui sta lavorando e iniziare a lavorare al immediatamente al blocco successivo, perché sa che gli altri nodi inizieranno a costruire sul blocco appena ricevuto.

1.4.2 Proof of Stake (PoS)

Il modello *proof of stake* è basato sul principio che più stake[3] l'utente ha investito nel sistema più è interessato al mantenere operativo il sistema stesso, o in altre parole più l'utente ha investito nel sistema più è disincentivato dal volerlo sovvertire. Lo stake è solitamente la quantità di criptovaluta che l'utente ha impegnato nel sistema (con diverse modalità, come ad esempio bloccando i fondi con una transazione speciale, inviandoli ad un certo indirizzo o possedendoli in speciali wallet). Uno volta nello stake, la criptovaluta generalmente non può più essere spesa. Le blockchain proof of stake usano il quantitativo di stake posseduto dall'utente come fattore determinante per la decidere chi può pubblicare nuovi blocchi. La facilità con cui un utente può

[3] *stake* è vagamente traducibile in italiano come "posta in gioco"

pubblicare blocchi è legata al rapporto tra stake investito dall'utente e stake investito in tutta la blockchain. A differenza della proof of work, con la proof of stake non è necessario effettuare computazioni onerose (a livello di tempo, elettricità e capacità di calcolo).

Poiché questo modello usa meno risorse, alcune blockchain decidono di non generare nuova criptovaluta nel tempo. Questi sistemi sono progettati in modo tale che tutta la criptovaluta circolante sia già distribuita tra gli utenti. In questi sistemi solitamente la ricompensa per la pubblicazione del blocco deriva solo dalle commissioni sulle transazioni.

1.4.3 Practical Byzantine Fault Tolerance (pBFT)

Il modello *practical byzantine fault tolerance* fornisce una pratica replica della macchina a stati bizantina in grado di funzionare anche in presenza di difetti in nodi (malevoli) indipendent e messaggi manipolati propagati da nodi specifici e indipendenti. I nodi nel modello pBFT sono ordinati sequenzialmente, dove uno di questi assume il ruolo di leader e gli il ruolo di nodi di backup. Tutti i nodi del sistema comunicano tra loro con l'obiettivo che tutti i nodi onesti riescano a raggiungere un consenso sullo stato del sistema. Per fare ciò devono dimostrare che i messaggi scambiati provengano da uno specifico nodo e devono verificare che i messaggi non siano stati modificati durante la trasmissione. Per il corretto funzionamento di pBFT il numero di nodi malevoli non deve essere uguale o superiore a un terzo rispetto a tutti i nodi del sistema in una data finestra di vulnerabilità. In pBFT la sicurezza del modello cresce al crescere dei partecipanti. Nel protocollo pBFT ogni round (chiamato *view*) è diviso in quattro fasi:

1. Un client invia una richiesta per una operazione di servizio al nodo leader.

2. Il leader fa il broadcast della richiesta a tutti i nodi di backup.

3. Il nodo esegue l'operazione richiesta e invia una conferma al client.

4. Il client attende $f + 1$[4] risposte dai nodi con lo stesso risultato. Tale risultato è considerato il risultato dell'operazione.

Il nodo leader è cambiato ad ogni view. Il leader può anche essere sostituito con un protocollo chiamato *view change* se passa un certo tempo senza che questo effettui il broadcast della richiesta del client. Inoltre, una maggioranza qualificata di nodi onesti può sostituire il leader se questo è malevolo con il nodo successivo nella lista.

1.4.4 Federated Byzantine Agreement (FBA)

Il modello *federated byzantine agreement* si basa sull'uso di *quorum* e di *quorum slice*. Un quorum è un insieme di nodi sufficiente per raggiungere un consenso. Un quorum slice è un sottoinsieme di un quorum in grado di convincere un particolare nodo ad un consenso. Ogni nodo può decidere autonomamente a quale quorum slice partecipare. Inoltre, può anche partecipare a più quorum slice contemporaneamente. L'obiettivo è quello di avere sovrapposizioni tra quorum, in modo che l'unione dei diversi quorum copra l'intero sistema. Quando i quorum non si intersecano, nel sistema si vengono a formare "quorum disgiunti". Questa è una situazione da evitare, poiché quorum disgiunti possono indipendentemente e simultaneamente raggiungere un accordo su transazioni contraddittorie, minando il consenso generale del sistema.

1.4.5 Round Robin

Il modello Round Robin è usato da alcune blockchain permissioned. In questo modello i nodi a turno acquisiscono il diritto di pubblicare blocchi. Il consenso Round Robin ha una lunga storia nelle architettura distribuite. Per gestire le situazioni dove

[4]dove f rappresenta il numero massimo di potenziali nodi difettosi

un nodo non può pubblicare un blocco quando è il suo turno, questi sistemi possono usare un tempo limite al diritto di pubblicazione. Questo modello garantisce che nessun nodo possa pubblicare la maggior parte dei blocchi.

I vantaggi sono l'approccio diretto, l'assenza di puzzle crittografici e il ridotto consumo di energia. Visto che è richiesta una fiducia tra i nodi, il modello Round Robin non funziona bene nelle blockchain permissionless. Questo perché attori malevoli possono continuamente aggiungere nodi per aumentare la probabilità di pubblicare blocchi, fino ad arrivare a sovvertire il corretto funzionamento della rete blockchain (*sybil attack*).

1.4.6 Proof of Authority/Proof of Identity

Il modello di consenso *Proof of Authority* (anche noto come *Proof of Identity*) si basa sulla fiducia parziale tra i nodi. L'identità dei nodi deve essere collegata con un soggetto del mondo reale; questo collegamento deve essere dimostrabile e verificabile all'interno della blockchain. L'idea è che il noto stia mettendo in gioco la sua reputazione quando pubblica blocchi. Il nodo può perdere la sua reputazione effettuando azioni in disaccordo con gli utenti della blockchain e allo stesso tempo può guadagnare reputazione agendo in modo concorde con gli altri utenti. Più è bassa la reputazione, minore è la probabilità di riuscire a pubblicare un blocco. Di conseguenza è nell'interesse del nodo mantenere una reputazione alta. Questo algoritmo è applicabile esclusivamente alla blockchain permissioned con un alto livello di fiducia.

1.4.7 Proof of Elapsed Time (PoET)

Nel modello *Proof of Elapsed Time* ogni nodo, quando vuole pubblicare un blocco, richiede un tempo di attesa casuale da una sorgente sicura hardware presente all'interno del proprio elaboratore. Il nodo ottiene il tempo casuale e va in idle per

quella durata. Dopo essersi risvegliato, il nodo generà un blocco e informa gli altri nodi del nuovo blocco; tutti gli altri nodi ancora in idle smettono di aspettare e l'intero processo riparte. Questo modello richiede la certezza dell'impiego di un tempo casuale, poiché se così non fosse un nodo malevolo potrebbe attendere sempre il tempo minimo e quindi dominare il sistema. Questo modello inoltre richiede la certezza che il nodo attenda realmente tutto il tempo assegnatogli senza partire in anticipo. Questi requisiti sono soddisfatti dall'eseguire il software in un ambiente di esecuzione affidabile presente in alcuni processori (come ad esempio Intel Software Guard Extensions [5], ARM TrustZone [6], AMD Platform Security Processor [7]).

[5]https://software.intel.com/en-us/sgx

[6]https://www.arm.com/products/silicon-ip-security

[7]https://www.amd.com/en/technologies/security

1.4.8 Tabella di comparazione

Nome	Obiettivi	Vantaggi	Svantaggi	Domini	Implementazioni
Proof of Work	Fornire una barriera alla pubblicazione dei blocchi attraverso un puzzle computazionalmente oneroso da risolvere per consentire transazioni tra partecipanti non fidati.	Difficile effettuare attacchi denial of service inviando blocchi malformati nella rete. Chiunque disponga di hardware può provare a risolvere il puzzle.	Computazionalmente onerosa (per progettazione), elevato consumo di energia, corsa all'armamento hardware. Potenziale attacco del 51% se si ottiene abbastanza potenza di calcolo.	Criptovalute permissionless.	Bitcoin, Ethereum, e molti altri
Proof of Stake	Consentire una barriera meno onerosa computazionalmente alla pubblicazione dei blocchi, ma consentire comunque transazioni tra partecipanti non fidati.	Meno computazionalmente onerosa della PoW. Aperta a chiunque voglia ottenere stake. Gli stakeholder controllano il sistema.	Nulla impedisce la formazione di un pool di stakeholder per la creazione di un potere centralizzato. Potenziale attacco 51% se si ottiene abbastanza stake. Gli stakeholder controllano il sistema.	Criptovalute permissionless.	Ethereum Casper, Krypton
pBFT	Replicare la macchina a stati bizantina distribuita per contrastare attacchi.	Efficienza energetica. Velocità. Uniformità di ricompensa per i nodi.	Sybil attack. Scalabilità	Sistemi permissioned.	Hyperledger Fabric, Zilliqa
FBA	Coprire l'intera rete con quorum sovrapposti per ridurre le interazioni necessarie per raggiungere il consenso.	Sicurezza asintotica Velocità	Possibili partizionamenti della rete.	Criptovalute permissionless.	Stellar, Ripple
Round Robin	Fornire un sistema per la pubblicazione di blocchi tra nodi approvati/fidati.	Computazionalmente non oneroso. Immediato da comprendere.	Richiede molta fiducia tra i nodi.	Sistemi permissioned.	MultiChain

Nome	Obiettivi	Vantaggi	Svantaggi	Domini	Implementazioni
Proof of Authority/Identity	Creare un processo di consenso centralizzato per minimizzare la velocità di creazione e validazione dei blocchi.	Tempi di conferma ridotti. Consente una velocità di produzione di blocchi dinamica. Può essere usato in sidechain a blockchain che usano altri modelli di consenso.	Si basa sull'assunzione che il nodo di validazione corrente non sia stato compromesso. Porta a un point of failure centralizzato. La reputazione di un nodo può essere compromessa in qualsiasi momento.	Sistemi permissioned. Sistemi ibridi (sidechain).	Ethereum Kovan testnet, POA Chain
Proof of Elapsed Time	Consentire un modello di consenso economico per la blockchain, a spese della maggior garanzia di sicurezza associata alla Pow.	Meno computazionalmente onerosa di Pow.	Necessita di hardware speciale. Assume che l'orologio hardware non sia compromesso. A causa di vincoli minimi di latenza, un vero sincronismo è essenzialmente impossibile nei sistemi distribuiti [23].	Reti permissioned.	Hyperledger Sawtooth

1.5 Smart Contract

Lo Smart Contract è uno strumento che ha un forte legame con la blockchain e che ha un ruolo chiave nelle applicazioni enterprise e non solo. Ciò nonostante, non vi è ancora un chiaro consenso sulla definizione di Smart Contract.

L'idea di Smart Contract risale a ben prima della nascita della Blockchain. È stata infatti introdotta per la prima volta nel 1994 da Nick Szabo, per poi essere formalizzato da lui stesso nel 1997 [45]. Secondo la definizione originale

> *Gli smart contract combinano protocolli con interfacce utente al fine di formalizzare e rendere sicure le relazioni su reti di computer. Obiettivi e principi per la progettazione di questi sistemi derivano da principi giuridici, dalla teoria economica e da teorie di protocolli affidabili e sicuri.*

Più recentemente sono emerse altre definizioni. In *Making sense of blockchain smart contracts* [22] Stark afferma che il termine Smart Contract ha assunto due diversi significati:

1. *Smart Contract Code*: opera nel mondo naturale, coinvolgendo l'uso di agenti software che tipicamente, ma non necessariamente, operano su un ledger condiviso. La parola contratto in questo senso indica che gli agenti software devono sottostare ad alcuni vincoli e possono prendere il controllo di alcuni asset in un ledger condiviso.

2. *Smart Legal Contract*: Il secondo significato si focalizza su come i contratti legali possono essere espressi ed eseguiti nel software. Ciò comprende quindi aspetti operativi, questioni relative a come i contratti legali sono scritti e come la prosa legale deve essere interpretata. Ci sono diverse idee e progetti incentrati su questi aspetti come il *Ricardian Contract* [20], *CommonAccord* [10] e *Legalese* [25].

Partendo dall'analisi di Spark, Clack et al. [9] propongono una definizione di più alto livello che include entrambi i significati di cui sopra, basata sui temi dell'automazione e dell'applicabilità:

> *Uno smart contract è un accordo automatizzabile e applicabile. Automatizzabile da computer, sebbene alcune parti possano richiedere input e controllo umani. Applicabile sia attraverso l'obbligatorietà legale di diritti e doveri o tramite l'esecuzione di codice informatico a prova di manomissione.*

Questa definizione è sufficientemente astratta da includere sia lo *smart legal contract* (dove l'accordo è un accordo legale, che è in grado di essere eseguito automaticamente da un software) e lo *smart contract code* (che potrebbe non essere necessariamente legato ad un accordo legale formale, ma che comunque viene eseguito automaticamente).

1.6 Vulnerabilità negli Smart Contract

Come abbiamo visto gli Smart Contract permettono l'esecuzione di codice arbitrario sulle blockchain, permettendo interazioni automatizzate tra diversi attori in un contesto *trustless* senza ricorrere all'utilizzo di terze parti fidate.

Gli Smart Contract si ritrovano a gestire diversi miliardi di USD in criptomoneta, attirando quindi le attenzioni di malintenzionati. In questa sezione analizzeremo diverse vulnerabilità emerse nel corso del tempo e mostreremo alcuni strumenti in grado di aiutarci nello sviluppo di Smart Contract sicuri.

1.6.1 Classificazione delle vulnerabilità

Secondo Nikolic et al. [32], gli Smart Contract vulnerabili possono essere divisi in tre categorie.

Prodigal Contracts Esistono alcune categorie di destinatari tipici delle transazioni effettuate dagli smart contracts. I contratti spesso restituiscono fondi ai proprietari, agli indirizzi che hanno inviato criptomoneta ad essi in passato (ad es. nelle lotterie) o a indirizzi per i quali vi è una ragione specifica (ad es. invio di una ricompensa). I **prodigal contracts** sono contratti che inviano fondi ad indirizzi arbitrari che non rientrano nelle categorie di cui sopra.

Suicidal Contracts I contratti spesso hanno una opzione di sicurezza che permette al proprietario o a un account fidato di uccidere il contratto in situazioni di emergenza, come un attacco o un malfunzionamento. Se questa funzionalità non è correttamente gestita un contratto potrebbe essere ucciso da un qualunque account arbitrario. Se è così, allora il contratto rientra nella categoria dei **suicidal contracts**.

Greedy Contracts I **greedy contracts** sono i contratti che rimangono vivi e detengono criptomoneta per un tempo indefinito e che non permettono il riscatto dei fondi contenuti in alcuna circostanza.

1.6.2 Strumenti per l'individuazione di vulnerabilità

Una buona pratica da seguire per sviluppare Smart Contracts sicuri è rimanere aggiornati sulle vulnerabilità note. Lo Smart Contract Weakness Classification Registry [42] fornisce un elenco continuamente aggiornato di vulnerabilità e anti-pattern noti insieme ad esempi di casi reali.

Strumenti di analisi automatizzata

Esistono diversi strumenti in grado di operare un'analisi per rilevare le vulnerabilità presenti negli Smart Contracts. Ne proponiamo un paio tra quelli che operano con

gli Smart Contract di Ethereum.

SECURIFY Tool: [40]. Paper: [46]. Si tratta di uno strumento per l'analisi statica di Smart Contract scritti in Solidity. È in grado di dimostrare che i comportamenti degli Smart Contract sono sicuri/non sicuri rispetto a determinate proprietà. L'analisi di Securify consiste in due passaggi. Per prima cosa analizza simbolicamente il grafo delle dipendenze del contratto per estrarre informazioni semantiche dal codice. Quindi usa modelli di conformità e violazione in grado di individuare eventuali criticità.

MAIAN Tool: [26]. Paper: [32] È uno strumento di analisi in grado di individuare vulnerabilità direttamente dal bytecode di Ethereum Smart Contract, senza l'uso dei sorgenti. Si basa sullo studio delle *trace properties*, impiegando l'uso di analisi simbolica inter-procedurale e di un validatore.

1.7 Stellar Network

Stellar è un protocollo decentralizzato ed open source nato con lo scopo di poter trasferire denaro attraverso diversi paesi con costi di transazione irrisori. Il nome della criptovaluta scambiata è Lumen (XLM).

È stato creato nel 2014 da *Jed McCaleb* e *Joyce Kim*. Si tratta di un progetto fortemente ispirato da *Ripple* (di cui Jed McCaleb è co-fondatore). A differenza di quest'ultimo, Stellar si pone come obiettivo il facilitare lo scambio di denaro tra privati invece che tra banche. Si tratta ovviamente di una rete *permissionless* (si veda 1.3.1).

Il protocollo di consenso utilizzato è stato descritto nel 2015 nel paper *The stellar consensus protocol: A federated model for internet-level consensus* [28]. Rientra nella categoria di modelli di consenso Federated Byzantine Agreement (si veda 1.4.4).

1.7.1 Gli Smart Contracts in Stellar

Secondo la documentazione ufficiale [44], uno *Stellar Smart Contract* (SSC) è espresso come la composizione di transazioni connesse fra loro ed eseguite secondo certi vincoli. I vincoli che si possono utilizzare nella realizzazione di SSCs sono:

- *Multisignature (multifirma)* - Quali chiavi sono necessarie per autorizzare una certa transazione? Quali soggetti devono concordare in una certa circorstanza affinche si possano eseguire i passi?

La Multisignature è il concetto di richiedere firme di soggetti diversi per effettuare transazioni provenienti da un certo account. Attraverso pesi e soglie di firma, viene creata la rappresentazione del potere nelle firme.

- *Batching/Atomicity (batching/atomicità)* - Quali operazioni devono avvenire o tutte insieme o fallire? Cosa deve accadere per forzare il successo o il fallimento?

Il Batching è il concetto dell'includere più operazioni in un'unica transazione. L'atomicità è la garanzia che, data una serie di operazioni raggruppate in un'unica transazione, al momento dell'invio sul network se anche una sola operazione dovesse fallire, tutte le operazioni nella transazione fallirebbero.

- *Sequence (sequenza)* - In che ordine deve essere elaborata una serie di transazioni? Quali sono le limitazioni e le dipendenze?

Il concetto di sequenza è rappresentato sullo Stellar network attraverso il numero di sequenza. Utilizzando i numeri di sequenza nella manipolazione delle transazioni è possibile garantire che transazioni specifiche non possano essere eseguite nel momento in cui venisse inoltrata una transazione alternativa.

- *Time Bounds (limiti temporali)* - Quando è possibile elaborare una transazione?

I limiti di tempo sono vincoli sul periodo di tempo durante il quale una transazione è valida. L'utilizzo dei limiti di tempo consente di rappresentare i periodi di tempo in un SSC.

1.8 L'adozione della blockchain nelle industrie

Il successo o il fallimento di una tecnologia è molto spesso legato più a quanto riesce a penetrare nella società che alla bontà della tecnologia stessa. In questa sezione vedremo alcuni dati e statistiche che ci aiuteranno a capire quale è lo stato dell'arte per quanto riguarda l'adozione delle tecnologie blockchain nelle industrie e cosa aspettarci per il futuro.

1.8.1 Il panorama attuale

Il numero di portafogli blockchain presenti a livello mondiale al termine di settembre 2018 era pari a circa 28 milioni [5]. Dalla statistica in figura 1.1 vediamo che questo valore è in continua crescita, con un trend positivo dall'andamento esponenziale.

Nel Q4 del 2017 abbiamo assistito ad una crescita più decisa del numero di portafogli blockchain. La ragione è che in quel periodo vi è stata un'esplosione del prezzo delle criptovalute, la quale ha attirato molti utenti ad interessarsi e ad investire nel mondo blockchain [19].

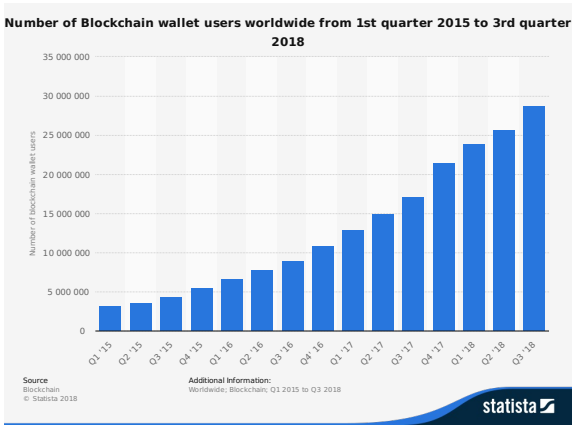


Figura 1.1: Numero di portafogli blockchain nel mondo [5].

In figura 1.2 vediamo i risultati di un sondaggio condotto a livello mondiale da *Deloitte* nell'aprile 2018. Alla domanda "Su quale modello blockchain stai concentrando le tue attività?" il 52% delle organizzazioni intervistate stanno concentrando i propri sforzi hanno risposto con *blockchain permissioned* [11].

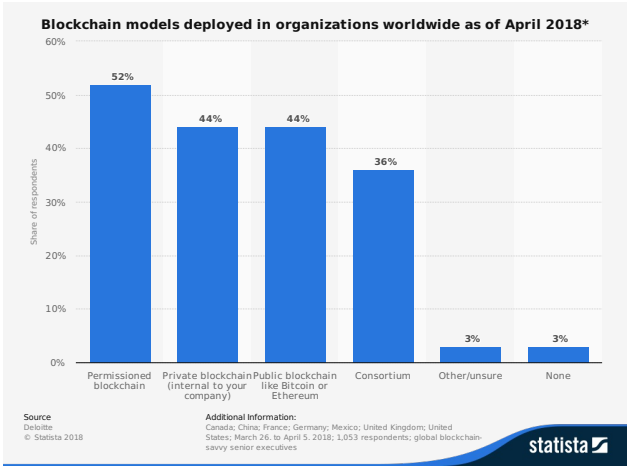


Figura 1.2: Modelli blockchain utilizzati dalle organizzazioni [11].

La blockchain è una tecnologia relativamente giovane. Come è lecito attendersi, le aziende non hanno ancora maturato una forte esperienza con questa tecnologia, ma ciononostante sono consapevoli che si tratta di una opportunità da seguire con

attenzione. Secondo *PwC* solo il 15% delle aziende U.S.A. attive nel mondo blockchain sono arrivate ad utilizzare la blockchain in produzione [35] e secondo *eft Supply Chain & Logistics Business Intelligence* il 69% delle organizzazioni intervistate ha dichiarato di stare investendo nel comprendere la tecnologia [14].

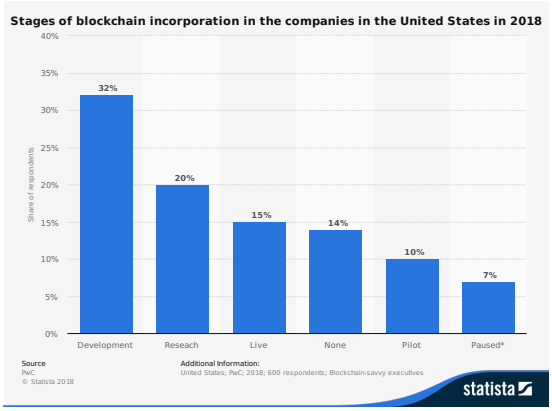


Figura 1.3: Livelli di adozione della blockchain nelle imprese U.S. [35].

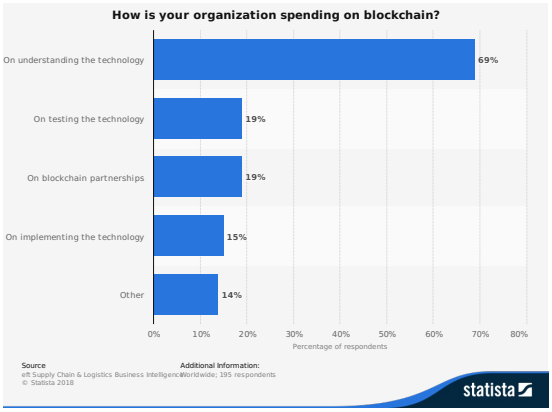


Figura 1.4: Sondaggio: in che modo la tua organizzazione sta investendo nella blockchain? [14].

1.8.2 L'evozione futura

In figura 1.5 viene presentata la spesa complessiva in soluzioni blockchain suddivisa per macroregioni geografiche tra il 2016 e il 2022. Nel 2022 la spesa per soluzioni blockchain a livello mondiale toccherà gli 11,65 miliardi di dollari e nei soli U.S.A viene proiettata a 4,2 miliardi di dollari [43].

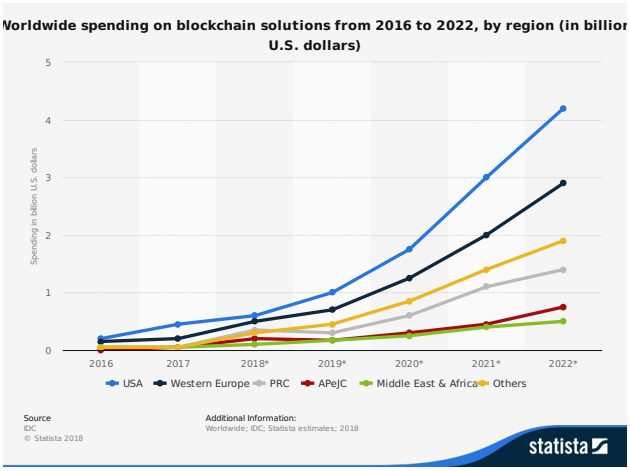


Figura 1.5: Spesa mondiale in soluzioni blockchain per regione (in miliardi di dollari U.S.) [43].

In figura 1.6 vengono mostrati i risultati di un sondaggio pubblicato in agosto 2018 in cui veniva chiesto quali sono i territori leader nello sviluppo di tecnologie blockchain. Circa il 18% dei partecipanti considera la Cina leader nel 2018. Alla domanda di considerare quale territorio sarebbe il leader nella tecnologia blockchain lo sviluppo dal 2021 al 2023, la percentuale di rispondenti che rispondono alla "Cina" è cresciuta al 30% [36].

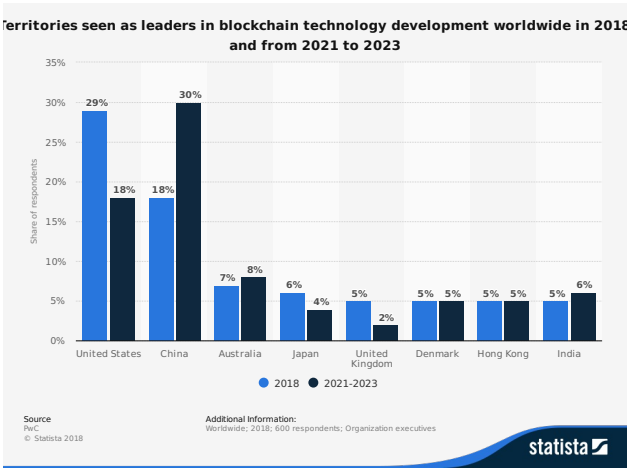


Figura 1.6: Territori mondiali considerati leader nello sviluppo di tecnologie blockchain nel 2018 e dal 2021 al 2023 [36].

1.8.3 Le aspettative delle imprese

Da un sondaggio condotto da *ABN Amro Economisch Bureau* nel 2017 [1] emerge che secondo i rivenditori olandesi la blockchain può essere uno strumento utile per velocizzare il proprio business. In particolare, il 77% degli intervistati ritiene che possa essere utile per velocizzare i pagamenti senza utilizzare banche come intermediari e il 58% crede che la blockchain possa velocizzare le operazioni logistiche riducendo la burocrazia cartacea e il coordinamento necessario tra le parti coinvolte. I risultati del sondaggio sono visibili in figura 1.7.

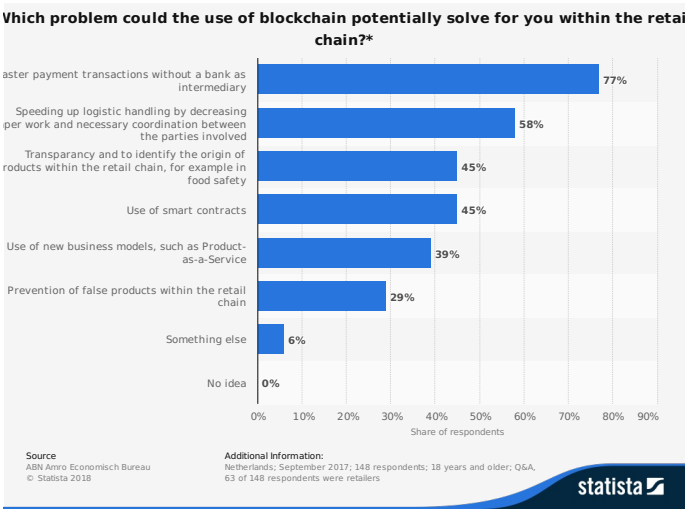


Figura 1.7: Problemi che può potenzialmente risolvere l'uso della blockchain nella catena di commercio [1].

In figura 1.8 vengono esposti i risultati di un sondaggio condotto tra banche, aziende FinTech e altri istituti finanziari nel 2017 [37]. L'obiettivo del sondaggio era quello di individuare quale fossero i casi d'uso di business della blockchain. È emerso che i più probabili casi d'uso commerciali della blockchain sono la creazione di una infrastruttura di pagamenti (67%) e finanza commerciale (61%). Da segnalare che il 44% degli intervistati ha dichiarato che la tecnologia blockchain può essere utile nella gestione dell'identità digitale.

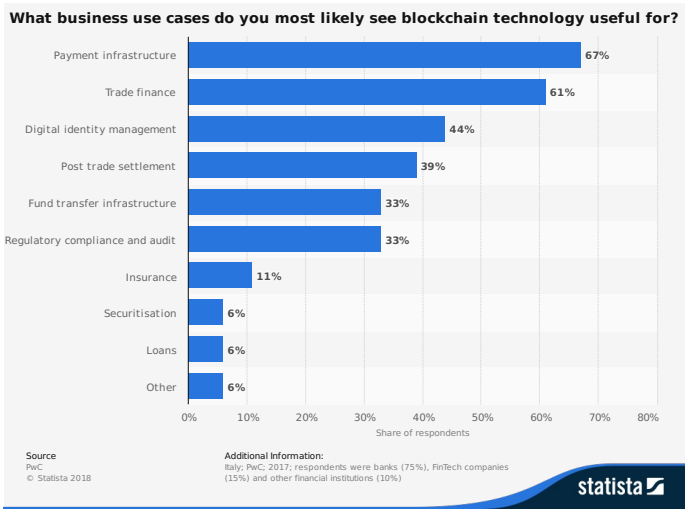


Figura 1.8: Casi d'uso commerciali per la blockchain [37].

L'aspetto che nel 2018 rappresenta il principale ostacolo alla adozione della tecnologia blockchain è l'incertezza normativa. Questo dato emerge dalla statistica condotta da *PwC* [34] e rappresentata in figura 1.9. Altre barriere rilevanti sono la mancanza di fiducia tra gli utenti e la capacità di riunire la rete.

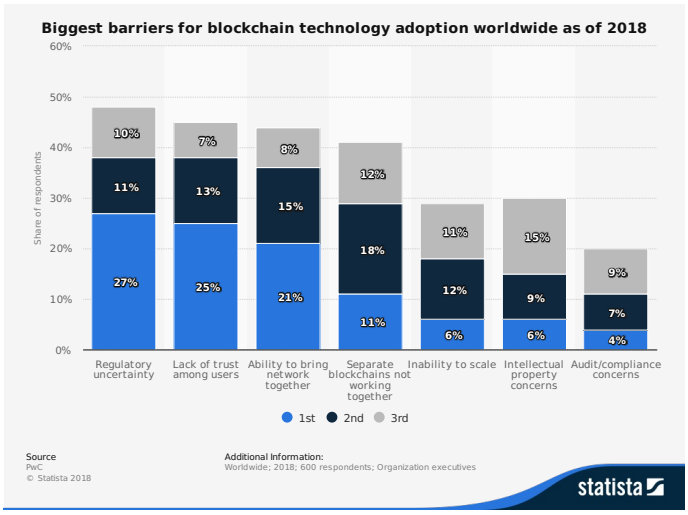


Figura 1.9: Barriere all'adozione della tecnologia blockchain [34].

Capitolo 2

Timed-Release Encryption

2.1 Cosa è la Timed-Release Encryption

La **Timed-Release Encryption** (TRE) è un metodo per cifrare un messaggio in modo che possa essere decifrato solo dopo che una certa deadline è stata raggiunta. Un avversario, anche se ha a disposizione una grande quantità di potenza di calcolo, non deve essere in grado di ottenere il messaggio prima della deadline. Inoltre, dopo la deadline i destinatari devono poter leggere il messaggio, indipendentemente dalla potenza di calcolo che hanno a disposizione e senza dover interagire con il mittente, con altri destinatari o con terze parti fidate.

Questo tipo di crittografia è molto difficile da ottenere con un modelli di computazione standard (come la macchina di Turing), perché questi non mettono a disposizione un "orologio del mondo reale" affidabile. Nel capitolo 3 proponiamo una soluzione che si appoggia sui DLT, i quali forniscono le buone proprietà di cui necessita la Timed-Release Encryption.

Il problema dell'*invio di un messaggio nel futuro* è stato proposto per la prima volta da May [27] nel 1993 e successivamente studiato da Rivest, Shamir and Wagner [39].

La Timed-Release Encryption ha diverse applicazioni nel mondo reale. Queste verranno discusse nel dettaglio nel capitolo 7.

2.2 Requisiti

Un buon sistema di TRE deve soddisfare i seguenti requisiti:

Certezza di pubblicazione Un sistema di TRE deve garantire con un buon grado di confidenza che al tempo τ il messaggio venga reso noto.

Segretezza del messaggio Un sistema di TRE deve garantire con un buon grado di confidenza che il messaggio rimanga segreto sino al tempo τ.

2.3 Implementare la Timed-Release Encryption

Le tecniche di implementazione della Timed-Release Encryption possono essere classificate in due categorie.

2.3.1 Time-Locked Puzzle

I Time-Locked puzzle sono stati introdotti per la prima volta da Merkle [29] e approfonditi da Bellare et. al. [3] e da Rivest et. al. [39]. L'idea che sta alla base è quella di cifrare il messaggio ad un certo istante di tempo t_0 in modo tale che ogni macchina (seriale o parallela) debba lavorare per un certo quantitativo di tempo Δ per risolvere il problema computazionale sottostante (puzzle). La somma tra t_0 e Δ è la deadline.

Sebbene rappresenti un approccio elegante nell'ambito della teoria della complessità computazionale, l'uso di *Time-Locked Puzzle* è difficilmente sostenibile nella pratica, poiché è poco flessibile e richiede l'uso di grandi quantità di risorse di calcolo.

Inoltre è necessario conoscere con precisione la potenza di calcolo a disposizione del risolutore per poter dimensionare la difficoltà del puzzle. Generalmente è possibile ottenere una buona stima di questo valore se ci si basa sull'hardware contemporaneo. È impossibile invece trovare una stima *future-proof*, poiché non sappiamo quali nuovi strumenti di calcolo verranno creati nel futuro. Ad ogni modo, anche se si avesse una stima affidabile a disposizione, con questo approccio sarebbe possibile garantire solo un limite inferiore sull'istante di pubblicazione del messaggio, dato che la macchina risolutrice potrebbe per vari motivi non iniziare immediatamente a lavorare all'istante t_0, potrebbe effettuare delle pause durante l'elaborazione o peggio ancora potrebbe non completare la risoluzione del problema.

2.3.2 Trusted agents (agenti fidati)

Questo approccio si basa su uno o più *agenti fidati* che rilasciano delle informazioni una volta che si è raggiunta la deadline. Questo può avvenire in maniera interattiva o non interattiva. Alcuni riferimenti [13, 6, 12, 8, 33, 18].

I vantaggi di questa strategia sono che non è richiesto che venga effettuata una computazione non-stop, il fatto che è possibile di rispettare con precisione la deadline e il fatto che non è richiesta una stima precisa della capacità di calcolo dei soggetti coinvolti. Tutto ciò ha un prezzo: gli agenti devono essere affidabili e devono essere attivi allo scoccare della deadline.

Capitolo 3

Il Protocollo

In questo capitolo verrà trattato il protocollo proposto per l'implementazione della Timed-Release Encryption sui DLT. Cercheremo di spiegare l'idea che sta alla base con l'ausilio di un esempio.

Rientra nella categoria degli approcci con *trusted agents* (si veda 2.3.2). L'idea è quella di distribuire agli agenti gli *share* del messaggio e sfruttare alcuni smart contract per fornire loro degli incentivi economici affinché pubblichino lo share loro assegnato una volta raggiunta la deadline. La blockchain viene utilizzata come orologio di riferimento per capire se la deadline è stata raggiunta.

3.1 Versione base

Immaginiamo che Carol abbia bisogno di Timed-Release Encryption su un certo messaggio x. Per farlo decide farsi aiutare da Alice. Quest'ultima impegna a conservare il messaggio e a renderlo pubblico solo dopo un certo istante di tempo τ. Diremo che Carol è il *client* e che Alice è l'*agent*.

Alice vuole essere retribuita per la conservazione di x. Viene quindi fissata una ricompensa *prize*. Chiaramente Alice vuole essere sicura di ottenere la ricompensa

se rispetta il suo impegno. Allo stesso tempo Carol vuole avere la certezza che Alice possa riscattare il premio solo se si comporta in maniera corretta. Carol inoltre non vuole che soggetti terzi partecipino all'accordo, perché desidera che sia x sia l'accordo stesso rimangano segreti. Per ottenere ciò decidono di usare uno smart contract.

Inizializzazione Nella fase iniziale Carol comunica il messaggio x ad Alice. Allo stesso tempo invia allo smart contract una certa cifra in criptomoneta che corrispone alla somma tra il *prize* da corrispondere ad Alice e un *pawn* che le verrà restituito al termine delle operazioni, un hash crittografico del messaggio x, l'istante di tempo τ e una tolleranza δ.

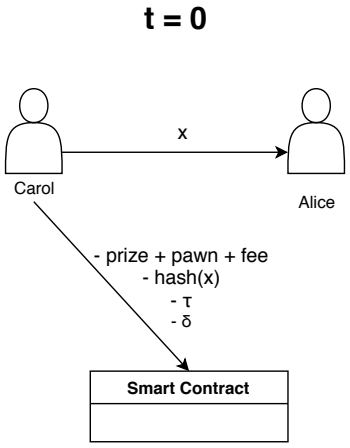


Figura 3.1: Inizializzazione (versione base).

Pubblicazione Alice conserva quindi x e lo rende noto al tempo t^* (con $\tau \leq t^* \leq \tau + \delta$)[1]. Per farlo invia allo smart contract il messaggio x, ed in cambio ottiene il suo *prize*. Allo stesso tempo, Alice riottiene il *pawn* che aveva versato in precedenza[2].

[1] Il δ serve a far si che Alice pubblichi puntualmente il messaggio. Se non ci fosse questo vincolo temporale potrebbe rilasciare il messaggio anche con molto ritardo rispetto a τ ed ottenere comunque la ricompensa.

[2] Il *pawn* è necessario per evitare alcuni comportamenti sleali da parte del client. La questione verrà approfondita nella sottosezione 6.1.5

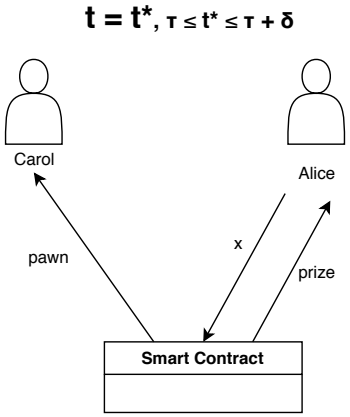


Figura 3.2: Pubblicazione del messaggio (versione base).

Lettura del messaggio Dopo il tempo t^*, ossia dopo che Alice ha inviato x allo smart contract, il messaggio diventa pubblico e chiunque può leggerlo.

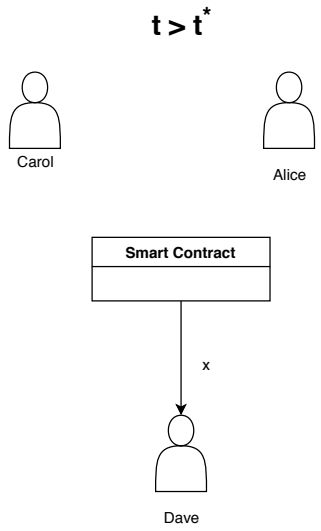


Figura 3.3: Lettura del messaggio (versione base).

Pubblicazione anticipata Cosa succede se Alice prova a riscattare il premio prima dell'istante τ? Semplicemente lo smart contract rifiuta la sua richiesta.

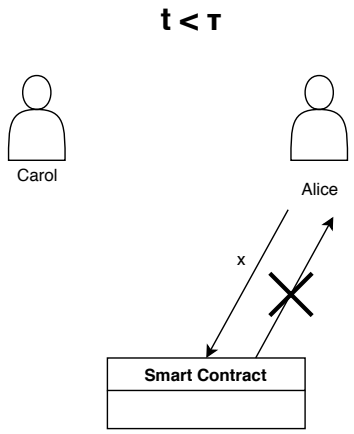


Figura 3.4: Tentativo di pubblicazione anticipata (versione base).

Leak E se Alice cede (volontariamente o a causa di un furto) il segreto ad Eve prima del tempo τ? In questo caso Eve può usare x per ottenere una ricompensa *counterprize* [3] Il riscatto del *counterprize* impedisce ad Alice di ottenere il suo premio. È evidente che l'interesse di Alice è quello di mantenere x segreto.

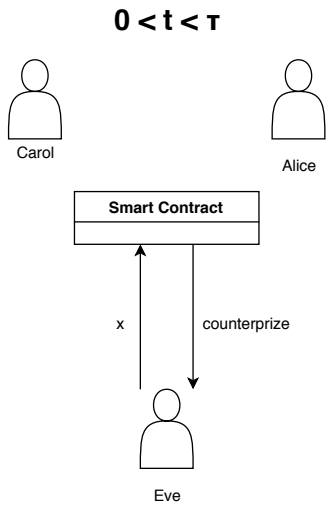


Figura 3.5: Riscatto del counterprize (versione base)

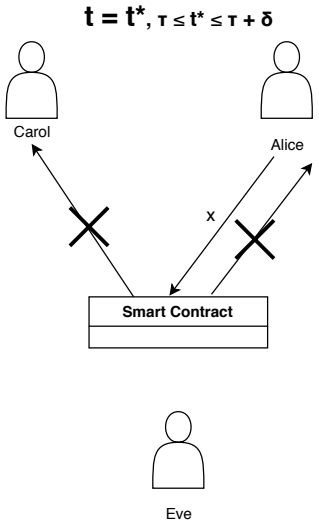


Figura 3.6: Impossibilità di riscatto del prize (versione base).

[3] dove *counterprize* $\ll$ *prize*. Le ragioni di questo vincolo sono spiegate al capitolo 5.

3.2 Versione avanzata

Facciamo notare che nella versione base Alice conosce sin dall'inizio il messaggio x, perché le è stato affidato nella prima fase del processo. Ma se Carol volesse che il messaggio rimanga segreto anche ad Alice? Per soddisfare questa condizione Alice ha bisogno di (almeno) un altro agente, Bob, e di un algoritmo di *secret sharing*[4].

Inizializzazione Alice, utilizzando un algoritmo di secret sharing, ottiene due share x_A e x_B. Invia questi share rispettivamente ad Alice e a Bob ed inizializza lo smart contract versando due *prize* e due *pawn*, inviando gli hash crittografici degli share, il tempo τ e la tolleranza δ.

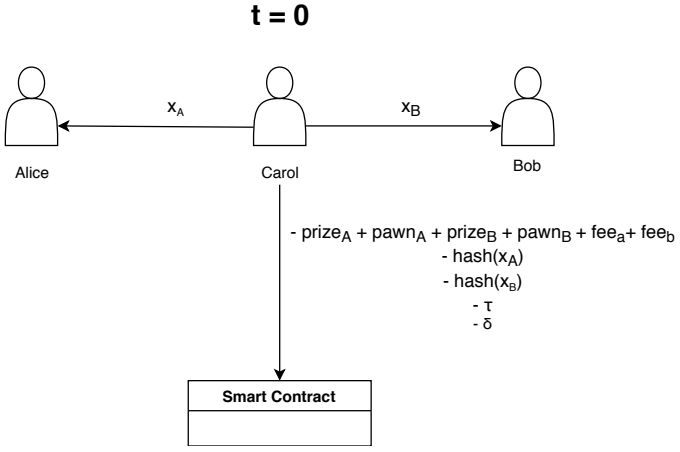


Figura 3.7: Inizializzazione (Versione avanzata).

Pubblicazione Alice e Bob pubblicano i loro share rispettivamente al tempo t_A^* e t_B^* (con $\tau \leq t_A^* \leq \tau + \delta$, $\tau \leq t_B^* \leq \tau + \delta$). In cambio ottengono i loro *prize*. Allo stesso tempo Alice riottiene i rispettivi *pawn* che aveva versato in precedenza.

[4]Un algoritmo di secret sharing è un algoritmo che permette di distribuire un certo segreto tra un gruppo di n partecipanti, ad ognuno dei quali viene assegnato uno *share*. Il segreto può essere ricostruito solo unendo un numero di share pari ad un fissato threshold t.

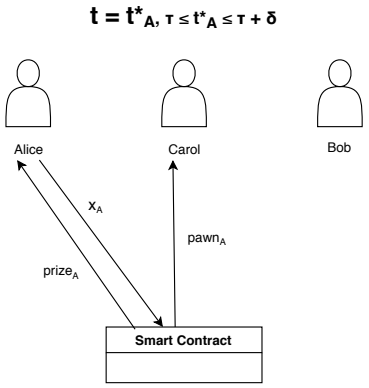


Figura 3.8: Pubblicazione dello share x_A (versione avanzata).

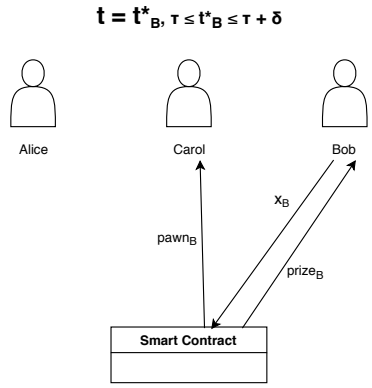


Figura 3.9: Pubblicazione dello share x_B (versione avanzata).

Accesso al messaggio Dopo il tempo $t^* = max(t_A^*, t_B^*)$, ossia dopo che sia Alice sia Bob hanno inviato x_A ed x_B allo smart contract, chiunque può leggere gli share dallo smart contract e quindi ricostruire il messaggio x.

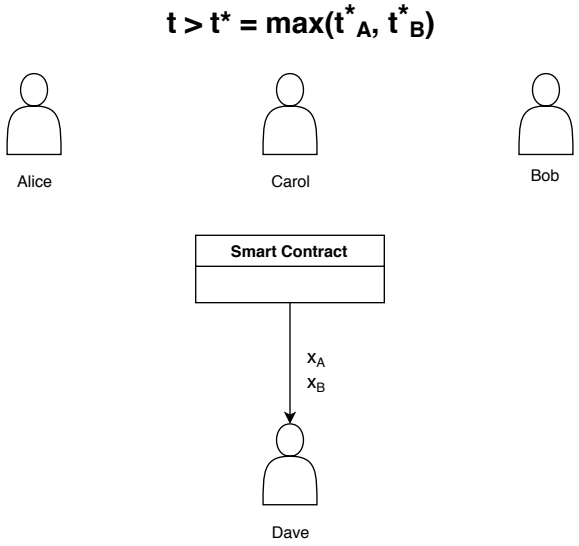


Figura 3.10: Lettura del messaggio (versione avanzata).

Leak Cosa succede se Bob cede il suo share ad Eve prima del tempo τ? Anche in questo caso Eve può usare x_B per ottenere una ricompensa *counterprize*. Il riscatto del *counterprize* impedisce a Bob di ottenere il suo premio. Da notare che se Alice

si comporta in maniera corretta, ossia se tiene segreto il suo share, è in grado di ottenere la sua ricompensa indipendentemente dal comportamento di Bob.

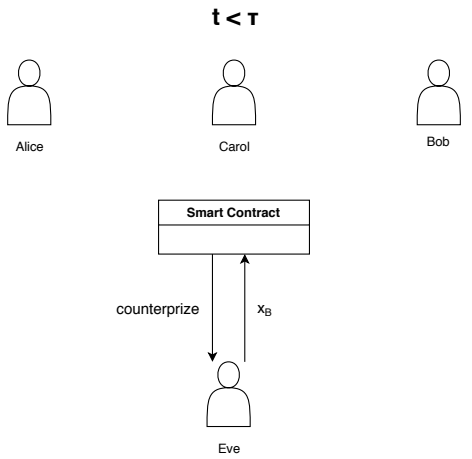


Figura 3.11: Riscatto del counterprize (versione avanzata).

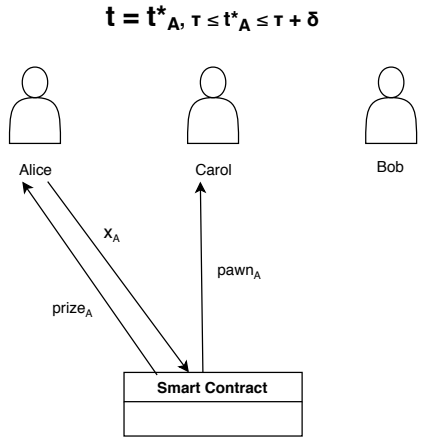


Figura 3.12: Riscatto di prize_A (versione avanzata).

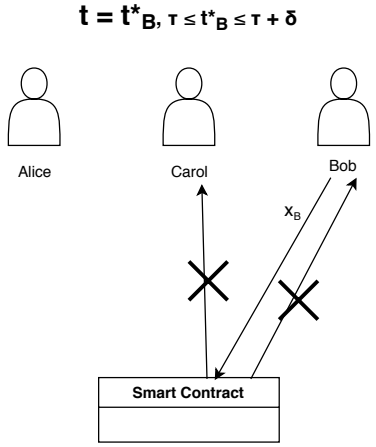


Figura 3.13: Impossibilità di riscatto di prize_B (versione avanzata).

Abbiamo visto un esempio con la partecipazione di due agenti, ma il protocollo può essere analogamente implementato con un numero qualsiasi N di agenti.

3.3 Varianti

3.3.1 Invio del messaggio solo a determinati attori

Dopo il tempo t^* il messaggio diventa pubblico a chiunque abbia accesso allo smart contract. In alcuni contesti questo potrebbe essere un problema, perché potremmo volere che il messaggio possa essere letto solo da determinati attori.

Una possibile soluzione è aggiungere un layer di cifratura simmetrica al messaggio prima di criptarlo con la TRE. La procedura è la seguente. Per prima cosa si cripta il messaggio m con cifratura simmetrica e chiave k ottenendo m'. Allo stesso tempo per ogni account destinatario designato $A_1, ..., A_n$ si generano $k_1, ..., k_n$, dove ogni k_i è k cifrato con la chiave pubblica del portafoglio A_i. A questo punto si genera un header $h = (k_1, ..., k_n)$. Infine si crea $x = (h, m')$ e lo si utilizza come segreto per il protocollo proposto. Una volta raggiunta la deadline x diventa pubblico, ma m no. I destinatari designati A_i possono leggere m estraendo k_i dall'header di x, ottenere k decriptandolo con la propria chiave privata e quindi decriptare m'.

3.3.2 Divisione di share in più livelli

La versione avanzata del protocollo proposto può essere estesa con una divisione su più livelli degli share distribuiti agli agenti. L'idea è quella di creare un *albero di distribuzione degli share*. Partendo dal messaggio x, che rappresenta il nodo radice, si ottengono gli share x_{1i}, rappresentati come nodi figli. A questo punto su ogni share x_{1i} si opera nuovamente il protocollo proposto, utilizzando questa volta x_{1i} come segreto. Si ottengono così gli share di secondo livello x_{2i}. Si può procedere iterativamente in questo modo fino a raggiungere la profondità desiderata. Gli share nei nodi foglia vengono poi assegnati agli agenti. Non è strettamente necessario creare un albero bilanciato.

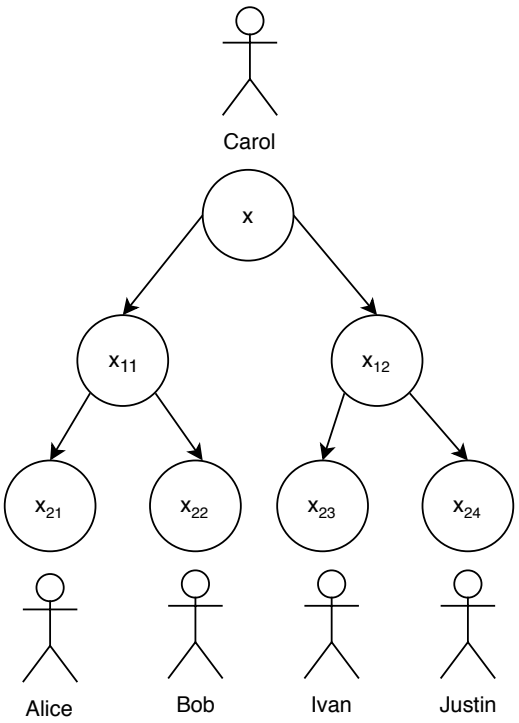


Figura 3.14: Un albero di distribuzione degli share.

Le scelte progettuali riguardo la struttura dell'albero di distribuzione degli share hanno un impatto sulla robustezza dell'implementazione del protocollo proposto. Per una analisi più approfondita rimandiamo al capitolo 5.

3.3.3 Ricompense intermedie

Immaginiamo che Carol abbia bisogno di Timed-Release Encryption su un certo messaggio x, e che la deadline τ sia molto avanti nel tempo. Immaginiamo inoltre che, fissata una ricompensa *prize*, l'agente Alice dia la sua disponibilità, ma a patto di non dover attendere fino al tempo τ per riceve l'intera ricompensa. Una soluzione è quella di prevedere delle ricompense intermedie. Per realizzarle si procede nel modo seguente.

Il messaggio x viene diviso in n share con threshold $t = n$. Per ogni share si applica la versione base del protocollo con deadline $\tau_1, \tau_2, ..., \tau_n = \tau$ tali che $\tau_1 < \tau_2 < ... < \tau_n = \tau$ e con rispettive ricompense $prize_1, prize_2, ..., prize_n$ tali che $prize_1 + prize_2 +, ... + prize_n = prize$. In questo modo l'agente Alice può eseguire delle pubblicazioni intermedie e ricevere le relative ricompense; allo stesso tempo il messaggio rimarrebbe segreto fino a quando non viene pubblicato l'ultimo share, ossia fino alla deadline τ. Questa metodologia può essere applicata analogamente anche alla versione avanzata.

3.3.4 Bonus per pubblicazione veloce dello share

Per come è progettato il protocollo, gli agenti possono pubblicare il loro share in un qualsiasi istante t^* compreso tra τ e $\tau + \delta$ e ricevere il *prize*. In alcuni casi si potrebbe desiderare avere un buon grado di confidenza sul fatto che la pubblicazione del messaggio avvenga in un istante t^* vicino alla deadline τ e, allo stesso tempo, avere un δ non troppo piccolo per permettere la pubblicazione del messaggio anche nel caso di agenti non puntuali. Per ottenere ciò, si possono incentivare gli agenti ad essere veloci premiandoli con dei bonus. Si può procedere in due modi.

Bonus a slot temporali Questa modalità consiste nel suddividere l'intervallo di tempo compreso tra τ e $\tau + \delta$ in N slot. Per ognuno di questi slot si assegna un bonus. Più lo slot è vicino alla deadline τ, più il bonus è elevato. Sulla base dello slot temporale in cui viene effettuato il rilascio dello share l'agente, oltre che al *prize*, riceve anche il bonus corrispondente.

Competizione per i bonus In questa modalità gli agenti competono per ottenere i bonus. A priori si stabilisce un bonus per il primo agente ad effettuare la

pubblicazione dello share, un bonus uguale o inferiore per il secondo agente e così via. Non è necessario che il numero dei bonus sia pari al numero degli agenti.

3.3.5 Cauzione per gli agenti

Per incentivare ulteriormente gli agenti a comportarsi in maniera corretta è possibile chiedere loro di versare una certa quantità di criptomoneta come cauzione nella fase di inizializzazione del protocollo. La cauzione viene restituita all'agente nel momento in cui effettua la pubblicazione dello share. Se, per un qualsiasi motivo, l'agente non riesce ad effettuare il rilascio, la cauzione è persa. Né il client, né un qualsiasi altro soggetto diverso dall'agente deve essere in grado di accedere alla cauzione. In caso contrario soggetti malevoli potrebbero sfruttare questa possibilità per mettere a segno delle frodi nei confronti degli agenti.

Capitolo 4

Proof of Concept

In questo capitolo costruiremo una soluzione sullo Stellar Network compatibile con il protocollo proposto al capitolo 3. Seguirà una Proof of Concept di quanto descritto.

Avremmo potuto proporre una implementazione anche su altri DLT, come ad esempio Ethereum. Abbiamo scelto Stellar perché ci da modo di dimostrare che per un'implementazione funzionante del protocollo proposto non è necessario un ambiente che offre degli smart contract "potenti" come ad esempio la *Ethereum Virtual Machine*, ma è sufficiente un supporto "debole" agli smart contract (si veda 1.7.1).

4.1 Un'implementazione sullo Stellar Network

Sia U_0 il *client*; siano $U_1, \dots, U_N$ gli *agenti* e siano rispettivamente $S_1, \dots, S_N$ i segreti assegnati agli agenti (share o messaggi).

Per prima cosa U_0 crea gli account $A_1, \dots, A_N$ e versa su questi account una cifra pari alla somma tra il *prize*, il *pawn* e il costo che dovrà sostenere per le commissioni delle transazioni che dovrà effettuare.

A questo punto per ogni account A_i il client crea N transazioni ognuna delle quali rivolta ad uno degli utenti $U_1, \ ... \ , U_N$, per un totale di $N \times N$ transazioni. Indichiamo con $T_{A_iU_j}$ la transazione proveniente dall'account A_i e diretta all'utente U_j. Per semplicità di notazione poniamo $T_{A_iU_j} = T_{ij}$.

Le transazioni sono così fatte:

- T_{ii} **Transazione Prize**
  - SEQUENCE_NUMBER = nextSequenceNumber(A_i) + 1
  - MIN_TIME = τ
  - MAX_TIME = $\tau + \delta$
  - OPERATIONS
    1. SEND *prize* TO U_i
    2. SEND *pawn* TO U_0
- T_{ij}, con $i \neq j$ **Transazione Counterprize**
  - SEQUENCE_NUMBER = nextSequenceNumber(A_i) + 1
  - MIN_TIME = *None*
  - MAX_TIME = τ
  - OPERATIONS
    1. SEND *counterprize* TO U_j

Facciamo notare che tutte queste transazioni hanno lo stesso sequence number. Ciò comporta che al più una di queste potrà essere eseguita.

Ora U_0 crea ulteriori N transazioni, una per ogni account, con queste caratteristiche:

- T_{i0} **Transazione d'inizializzazione**

- SEQUENCE_NUMBER = nextSequenceNumber(A_i)
- OPERATIONS
  1. SET *master weight* = 255
  2. SET *med threshold* = 2
  3. SET *high threshold* = 254
  4. APPEND PRE_AUTH_TX *hash(T_{ij})* WITH WEIGHT 1 (per ogni T_{ij}, con $1 \leq j \leq N$)
  5. APPEND HASHX_SIGNER *hash(S_i)* WITH WEIGHT 1
  6. SET *master weight* = 0

A U_0 non resta che inviare ad ogni utente U_i[1] il segreto S_i, le transazioni prize e counterprize T_{ij} e fare il submit delle transazioni d'inizializzazione T_{i0}.

Analizziamo nel dettaglio la transazione d'inizializzazione. Il *med threshold* è la soglia da raggiungere per effettuare l'invio di Lumen dall'account. Con l'operazione 2 lo abbiamo impostato a 2. Con le operazioni 4 abbiamo pre-autorizzato le transazioni T_{ij} con $1 \leq j \leq N$ con peso pari a 1. Con l'operazione 5 abbiamo aggiunto il segreto S_i tra le firme autorizzate, con peso pari a 1. Infine con le operazioni 3 e 6 abbiamo impedito all'utente U_0 (il detentore della masterkey) di effettuare alcun tipo di operazione dall'account A_i. L'operazione 1 ci è servita solo per non avere problemi nell'effettuare le operazioni successive.

Lo scenario finale è che l'account A_i è bloccato: neanche U_0, il detentore della master key, può apportarvi modifiche o prelevare fondi. Le uniche operazioni autorizzate sono le T_{ij}, ossia le transazioni per ottenere il *prize* o il *counterprize*. Per fare il submit

[1] con $1 \leq i \leq N$

di queste transazioni è inoltre necessario conoscere S_i, perché in questo modo si riesce a raggiungere il med threshold.

4.2 Limitazioni della soluzione proposta

4.2.1 Numero massimo di firme per transazione

Una limitazione imposta dallo Stellar Network è che con una singola transazione è possibile aggiungere al massimo 20 firme ad un account. Ciò comporta che se $N >= 20$ è necessario creare più di una transazione d'inizializzazione per ogni account. In generale, il numero delle transazioni d'inizializzazione necessarie è

$$\#T_{i0} = \left\lfloor \frac{N+1}{20} \right\rfloor$$

4.2.2 Riscatto dei *counterprize*

L'implementazione proposta si basa sul fatto che in Stellar è possibile pre-autorizzare delle transazioni. Uno dei problemi è che non è possibile pre-autorizzare delle transazioni con un destinatario generico, ma è necessario definirlo a priori. Nel nostro caso ciò comporta che i possibili destinatari dei *counterprize* devono essere decisi già nella fase di inizializzazione. Per il tipo di soluzione proposta non vi è un limite teorico per il numero di transazioni *counterprize* che si possono emettere, ma chiaramente non è sostenibile l'emissione di transazioni verso tutti i potenziali account Stellar. Si è deciso quindi di emettere transazioni *counterprize* riscattabili solo dagli agenti coinvolti.

Ciò comporta che solo gli agenti possono riscattare il *counterprize*. Questo potrebbe essere un problema, perché significa che l'agente non avrebbe il disincentivo economico nel cedere il proprio share ad un soggetto esterno. Di seguito proponiamo un protocollo in grado di risolvere questa situazione.

Immaginiamo che Eve ottenga lo share dall'agente Alice. Quello che può fare Eve è decidere di dividersi il *counterprize* con l'agente Bob. Per farlo Bob pre-autorizza due transazioni

- T_1
  - SEQUENCE_NUMBER = nextSequenceNumber(A_B) + 1
  - OPERATIONS
    1. SEND (*counterprize* / 2) TO U_j
    2. SET *master weight* = 1
- T_2
  - SEQUENCE_NUMBER = nextSequenceNumber(A_B) + 1
  - MIN_TIME = $\bar{t}$
  - MAX_TIME = *None*
  - OPERATIONS
    1. SET *master weight* = 1

e inizializza lo scambio con

- T_0
  - SEQUENCE_NUMBER = nextSequenceNumber(A_B)
  - OPERATIONS
    1. SEND *all* TO A'_B
    2. APPEND PRE_AUTH_TX *hash(T_1)* WITH WEIGHT 1
    3. APPEND PRE_AUTH_TX *hash(T_2)* WITH WEIGHT 1
    4. SET *master weight* = 0

Quello che avviene con la transazione T_0 e che Bob svuota l'account destinatario del *counterprize* A_B, si esclude dagli aventi diritto di firma su A_B e allo stesso tempo pre-autorizza le due transazioni T_1 e T_2.

La transazione T_1 invia ad Eve metà del *counterprize* e ripristina la firma di Bob su A_B. L'idea è che sia Eve a fare il submit di questa transazione. L'unico modo che ha per farlo è quello di fare il submit della transazione *counterprize*, in modo da avere sull'account A_B un deposito di fondi sufficienti.

Infine T_2 serve a tutelare Bob; se Eve non fa il submit di T_1 in un tempo ragionevole $\bar{t}$ Bob può usare T_2 per riprendere il possesso del suo account A_B.

4.3 Lo script

Di seguito uno script scritto in Python che applica quanto detto sopra. Si fa uso del *Python SDK per Stellar* e di *secretsharing di Blockstack*, un'implementazione open-source dello Shamir Secret Sharing [41].

Simuliamo la *versione avanzata*, con Carol che genera due share e li affida ad Alice e Bob. Simuliamo inoltre che Alice venga in possesso dello share di Bob, e quindi riesca ad ottenere il *counterprize* associato.

```
from stellar_base.keypair import Keypair
from stellar_base.address import Address
from stellar_base.builder import Builder
from stellar_base.transaction_envelope import TransactionEnvelope
                                    as Te
from stellar_base.stellarxdr.StellarXDR_type import TimeBounds
from stellar_base.horizon import horizon_testnet
from secretsharing import PlaintextToHexSecretSharer
import requests
import hashlib
```

```
import datetime
import time
import base64


def initialize_keypair(add_funds=True):
    kp = Keypair.random()
    if add_funds:
        publickey = kp.address().decode()
        url = 'https://friendbot.stellar.org/?addr=' + publickey
        requests.get(url)
    return kp


def extract_share_from_tx(tx):
    return base64.b64decode(
        tx['_embedded']['records'][0]['signatures'][0]).decode()


PRIZE = '100'
PAWN = '100'
COUNTERPRIZE = '1'
DELTA = 60  # seconds

x = 'Hello, World!'  # Carol's message
x_a, x_b = PlaintextToHexSecretSharer.split_secret(x, 2, 2)

hash_x_a = hashlib.sha256(x_a.encode()).digest()
hash_x_b = hashlib.sha256(x_b.encode()).digest()
horizon = horizon_testnet()

print('Initializing keypairs')
kp_alice = initialize_keypair()
```

```
kp_bob = initialize_keypair()
kp_carol = initialize_keypair()

print('Carol creates accounts a_1 and a_2')
kp_a_1 = initialize_keypair(add_funds=False)
kp_a_2 = initialize_keypair(add_funds=False)
kp_a_1_address = kp_a_1.address().decode()
kp_a_2_address = kp_a_2.address().decode()
kp_a_1_seed = kp_a_1.seed().decode()
kp_a_2_seed = kp_a_2.seed().decode()
accounts_builder = Builder(secret=kp_carol.seed().decode())
accounts_builder.append_create_account_op(
    kp_a_1_address, '202.50009')
accounts_builder.append_create_account_op(
    kp_a_2_address, '202.50009')
accounts_builder.sign()
response = accounts_builder.submit()
assert 'hash' in response

print('Initializing transactions')
address_a_1 = Address(address=kp_a_1_address)
address_a_2 = Address(address=kp_a_2_address)
address_a_1.get()
address_a_2.get()

starting_sequence_a_1 = int(address_a_1.sequence)
starting_sequence_a_2 = int(address_a_2.sequence)

tau = datetime.datetime.now() + datetime.timedelta(seconds=30)
tau_unix = int(time.mktime(tau.timetuple()))
tau_plus_delta = tau + datetime.timedelta(seconds=DELTA)
tau_plus_delta_unix = int(time.mktime(tau_plus_delta.timetuple()))
tx_timebound = TimeBounds(
```

```
    minTime=tau_unix, maxTime=tau_plus_delta_unix)
counter_tx_timebound = TimeBounds(minTime=0, maxTime=tau_unix)

builder_t_1_1 = Builder(secret=kp_a_1_seed,
                        sequence=starting_sequence_a_1+1)
builder_t_1_1.append_payment_op(
    kp_alice.address().decode(), PRIZE, 'XLM')
builder_t_1_1.append_payment_op(
    kp_carol.address().decode(), PAWN, 'XLM')
builder_t_1_1.add_time_bounds(tx_timebound)
t_1_1 = builder_t_1_1.gen_tx()
hash_t_1_1 = builder_t_1_1.gen_te().hash_meta()

builder_t_1_2 = Builder(secret=kp_a_1_seed,
                        sequence=starting_sequence_a_1+1)
builder_t_1_2.append_payment_op(
    kp_bob.address().decode(), COUNTERPRIZE, 'XLM')
builder_t_1_2.add_time_bounds(counter_tx_timebound)
t_1_2 = builder_t_1_2.gen_tx()
hash_t_1_2 = builder_t_1_2.gen_te().hash_meta()

builder_t_2_2 = Builder(secret=kp_a_2_seed,
                        sequence=starting_sequence_a_2+1)
builder_t_2_2.append_payment_op(
    kp_bob.address().decode(), PRIZE, 'XLM')
builder_t_2_2.append_payment_op(
    kp_carol.address().decode(), PAWN, 'XLM')
builder_t_2_2.add_time_bounds(tx_timebound)
t_2_2 = builder_t_2_2.gen_tx()
hash_t_2_2 = builder_t_2_2.gen_te().hash_meta()

builder_t_2_1 = Builder(secret=kp_a_2_seed,
                        sequence=starting_sequence_a_2+1)
```

```
builder_t_2_1.append_payment_op(
    kp_alice.address().decode(), COUNTERPRIZE, 'XLM')
builder_t_2_1.add_time_bounds(counter_tx_timebound)
t_2_1 = builder_t_2_1.gen_tx()
hash_t_2_1 = builder_t_2_1.gen_te().hash_meta()

builder_t_1_0 = Builder(secret=kp_a_1_seed)
builder_t_1_0.append_set_options_op(master_weight=255)
builder_t_1_0.append_set_options_op(med_threshold=2)
builder_t_1_0.append_set_options_op(high_threshold=254)
builder_t_1_0.append_pre_auth_tx_signer(hash_t_1_2, 1)
builder_t_1_0.append_pre_auth_tx_signer(hash_t_1_1, 1)
builder_t_1_0.append_hashx_signer(hash_x_a, 1)
builder_t_1_0.append_set_options_op(master_weight=0)
builder_t_1_0.sign()

builder_t_2_0 = Builder(secret=kp_a_2_seed)
builder_t_2_0.append_set_options_op(master_weight=255)
builder_t_2_0.append_set_options_op(med_threshold=2)
builder_t_2_0.append_set_options_op(high_threshold=254)
builder_t_2_0.append_pre_auth_tx_signer(hash_t_2_1, 1)
builder_t_2_0.append_pre_auth_tx_signer(hash_t_2_2, 1)
builder_t_2_0.append_hashx_signer(hash_x_b, 1)
builder_t_2_0.append_set_options_op(master_weight=0)
builder_t_2_0.sign()

print('Submitting t_1_0')
response = builder_t_1_0.submit()
assert 'hash' in response

print('Submitting t_2_0')
response = builder_t_2_0.submit()
assert 'hash' in response
```

```
print('At this point Carol cannot remove funds from a_1/a_2')
builder = Builder(secret=kp_a_1_seed)
builder.append_payment_op(kp_carol.address().decode(), 1)
builder.sign()
response = builder.submit()
assert 'hash' not in response

now = datetime.datetime.now()
print('Alice tries to submit t_1_1 before the deadline, but fails')
print(f'[deadline: { tau }; current time: { now }]')
envelope = Te(t_1_1, opts={})
response = horizon.submit(envelope.xdr())
assert 'hash' not in response

print('We suppose Bob leaks his secret. Alice can submit t_2_1')
envelope = Te(t_2_1, opts={})
envelope.sign_hashX(x_b)
response = horizon.submit(envelope.xdr())
assert 'hash' in response

print('Waiting for the deadline')
tts = (tau - datetime.datetime.now()).total_seconds()
margin = 5
time.sleep(tts + margin)

print('Now Alice can submit t_1_1')
envelope = Te(t_1_1, opts={})
envelope.sign_hashX(x_a)
response = horizon.submit(envelope.xdr())
assert 'hash' in response

print('Bob cannot submit t_2_2, because he leaked the secret')
```

```
envelope = Te(t_2_2, opts={})
envelope.sign_hashX(x_b)
response = horizon.submit(envelope.xdr())
assert 'hash' not in response

print('At this point, the secret is public')
params = {'limit': 1, 'order': 'desc'}
last_tx_a_1 = horizon.account_transactions(
    kp_a_1_address, params=params)
last_tx_a_2 = horizon.account_transactions(
    kp_a_2_address, params=params)
x_a_1 = extract_share_from_tx(last_tx_a_1)
x_a_2 = extract_share_from_tx(last_tx_a_2)
secret = PlaintextToHexSecretSharer.recover_secret([x_a_1, x_a_2])
assert secret == x
```

https://github.com/imcatta/stellar-timed-release-encryption-poc

Capitolo 5

Analisi di robustezza

In questo capitolo presenteremo una tecnica per valutare della robustezza delle implementazioni del protocollo proposto. L'obiettivo è quello di trovare la probabilità R_c che il segreto venga pubblicato una volta raggiunta la deadline (requisito di *certezza di pubblicazione*) e la probabilità R_s che il messaggio rimanga segreto fino alla deadline (requisito di *segretezza del messaggio*).

5.1 Terminologia

Numero di share Il numero di share n è uno dei parametri dell'algoritmo di secret sharing, utilizzato dalla versione avanzata del protocollo. Facciamo notare che la versione base può essere vista come un caso particolare della versione avanzata, dove $n = 1$.

Threshold per la ricostruzione del segreto Il threshold t rappresenta il numero minimo di share che sono necessari per la ricostruzione del segreto. Anche in questo caso la versione base può essere vista come un caso particolare della versione avanzata, dove $t = 1$.

Resistenza a smarrimenti La resistenza agli smarrimenti θ rappresenta il numero di agenti che possono non rendere noto il proprio share senza compromettere la pubblicazione del messaggio originale.

$$\theta = n - t$$

È interessante valutare questo valore in relazione al numero totale di share n.

$$\theta_r = \frac{n-t}{n}$$

$$\theta_{\%} = 100 \cdot \theta_r = 100 \cdot \frac{n-t}{n}$$

Resistenza a furti La resistenza ai furti γ è il numero di share che un soggetto deve riuscire ad ottenere dai agenti affinchè possa ricostruire il segreto prima del tempo τ. Questo numero è ovviamente pari al threshold.

$$\gamma = t$$

5.2 Tecnica di analisi

Per prima cosa è necessario assegnare ad ogni agente tre attributi con un valore compreso tra 0 e 1, assumendo che tutti gli agenti siano entità ben distinte e che non vi siano coalizioni tra agenti.

- P_s: la probabilità che l'agente smarrisca il segreto.
- P_c: la probabilità che l'agente ceda lo share a lui assegnato ad un altro soggetto.
- P_m: la probabilità che l'agente cerchi di corrompere altri agenti per ottenere i loro share.

A questo punto si rappresenta l'albero di distribuzione degli share. Partendo dai nodi foglia si calcolano gli attributi P_s, P_c e P_m per i rispettivi nodi padre. Si procede iterativamente in questo modo fino a quando non si raggiunge il nodo radice.

R_c è la probabilità che il messaggio venga pubblicato una volta raggiunta la deadline, ossia l'evento opposto allo smarrimento

$$R_c = 1 - P_c$$

R_s è la probabilità che il messaggio rimanga segreto, ossia uno meno la probabilità che venga compromesso da un attore esterno P_c sommata alla probabilità che venga compromesso da un agente P_m.

$$R_s = 1 - (P_c + P_m)$$

Di seguito alcuni esempi.

Albero di distribuzione con profondità 1 Sia Carol il client e siano Alice, Bob e Dave gli agenti. Siano $P_s^A = 0{,}2 \quad P_c^A = 0{,}25 \quad P_m^A = 0{,}001$ gli attributi di Alice; siano $P_s^B = 0{,}35 \quad P_c^B = 0{,}35 \quad P_m^B = 0{,}0005$ gli attributi di Bob; siano $P_s^C = 0{,}005 \quad P_c^C = 0{,}005 \quad P_m^C = 0{,}12$ gli attributi di Dave. Ipotizziamo che Carol usi la versione avanzata del protocollo proposto con numero di share $n = 3$ e threshold $t = 2$.

Abbiamo che la resistenza agli smarrimenti è $\gamma = 1$, quindi per calcolare P_s del nodo radice dobbiamo trovare la probabilità che vengano smarriti almeno $\gamma + 1$ share.

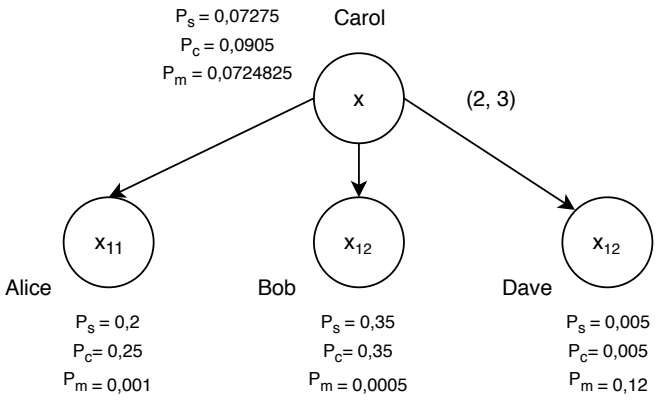
$$P_s = P_s^A \cdot P_s^B + P_s^A \cdot P_s^C + P_s^B \cdot P_s^C = 0{,}07275$$

La resistenza ai furti è $\theta = 1$. Analogamente a prima dobbiamo calcolare la probabilità che almeno $\theta + 1$ share vengano sottratti agli agenti.

$$P_c = P_c^A \cdot P_c^B + P_c^A \cdot P_c^C + P_c^B \cdot P_c^C = 0{,}0905$$

Ci resta da trovare P_m. Per ogni agente dobbiamo calcolare la probabilità che sia malintenzionato e che riesca a recuperare gli altri $n-1$ share necessari per ottenere il segreto.

$$P_m = P_m^A \cdot P_c^B + P_m^A \cdot P_c^C + P_m^B \cdot P_c^A + P_m^B \cdot P_c^C + P_m^C \cdot P_c^A + P_m^C \cdot P_c^B + = 0{,}0724825$$



Concludiamo trovando R_c e R_s.

$$R_c = 1 - 0{,}07275 = 0{,}92725$$
$$R_s = 1 - 0{,}0905 - 0{,}0724825 = 0{,}8370175$$

Albero di distribuzione bilanciato di profondità 2 Immaginiamo di avere a disposizione quattro agenti. Per tre di questi vale $P_s = 0{,}1 \quad P_c = 0{,}3 \quad P_m = 0{,}02$ mentre per il quarto abbiamo $P_s = 0{,}02 \quad P_c = 0{,}02 \quad P_m = 0{,}005$. Immaginiamo inoltre di utilizzare un albero di distribuzione simmetrico di profondità 2. Come parametri per l'algoritmo di secret sharing usiamo $t = 2$ e $n = 2$ in tutti i casi. La situazione è rappresentata in figura 5.1.

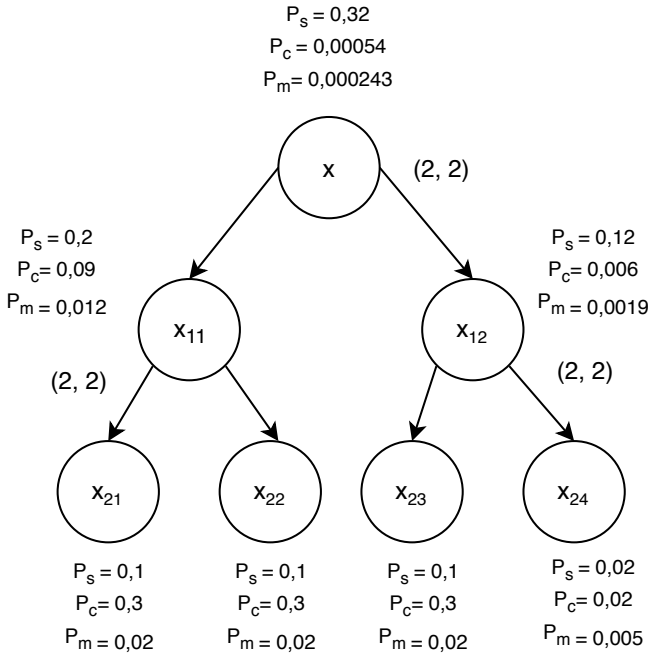


Figura 5.1: Albero di distribuzione bilanciato di profondità 2.

Calcoliamo gli attributi di x_{11}. Abbiamo $t = 2$ e $n = 2$. La resistenza agli smarrimenti è $\gamma = 0$, di conseguenza la P_s di x_{11} è data dalla somma delle P_s dei nodi figli.

$$P_s = 0{,}1 + 0{,}1 = 0{,}2$$

Per poter ottenere lo share x_{11} un attaccante esterno deve riuscire ad ottenere sia x_{21} sia x_{22}. Pertanto P_c è dato dal prodotto dei P_c dei nodi figli.

$$P_c = 0{,}3 \cdot 0{,}3 = 0{,}09$$

Se invece è uno degli agenti a voler ottenere x_{11} è sufficiente che ottenga lo share dell'altro agente. Quindi P_m è data dalla somma delle probabilità che si abbia sia un agente scorretto sia un agente corrotto.

$$P_m = (0{,}02 \cdot 0{,}3) + (0{,}02 \cdot 0{,}3) = 0{,}0012$$

Per quanto riguarda gli attributi di x_{12} abbiamo anche qui $t = 2$ e $n = 2$. Il ragionamento è analogo a quello di cui sopra. Ci limitiamo a riportare i calcoli.

$$P_s = 0{,}1 + 0{,}02 = 0{,}12$$
$$P_c = 0{,}3 \cdot 0{,}02 = 0{,}006$$
$$P_m = (0{,}02 \cdot 0{,}02) + (0{,}005 \cdot 0{,}3) = 0{,}0019$$

A questo punto dobbiamo trovare gli attributi di x. Si ha $t = 2$ e $n = 2$, anche qui il ragionamento è analogo a prima.

$$P_s = 0{,}2 + 0{,}12 = 0{,}32$$
$$P_c = 0{,}09 \cdot 0{,}006 = 0{,}00054$$
$$P_m = (0{,}012 \cdot 0{,}006) + (0{,}0019 \cdot 0{,}09) = 0{,}000243$$

Abbiamo quindi che

$$R_c = 1 - 0{,}32 = 0{,}68$$
$$R_s = 1 - 0{,}00054 - 0{,}000243 = 0{,}999217$$

Albero di distribuzione con ridondanza Analogamente al caso precedente immaginiamo di avere a disposizione quattro agenti. Per tre di questi vale $P_s = 0{,}1$ $P_c = 0{,}3$ $P_m = 0{,}02$ mentre per il quarto abbiamo $P_s = 0{,}02$ $P_c = 0{,}02$ $P_m = 0{,}005$. Immaginiamo inoltre di utilizzare un albero di distribuzione simmetrico di profondità 2. A differenza del caso precedente però utilizziamo $t = 1$ come threshold per gli share di secondo livello. Facciamo notare che $t = 1$ significa che basta un solo share per ricostruire il messaggio originale, ossia lo share è il messaggio originale. Di fatto possiamo vedere questa situazione come una distribuzione degli share di primo livello ridondanti. La situazione è rappresentata in figura 5.2.

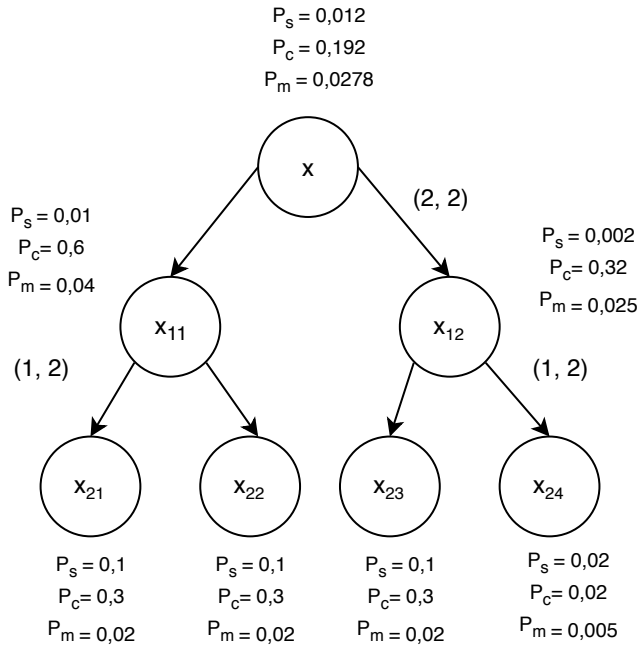


Figura 5.2: Albero di distribuzione con ridondanza.

Calcoliamo gli attributi di x_{11}. Abbiamo $t = 1$ e $n = 2$. Di fatto x_{11} è distribuito due volte, quindi x_{11} viene smarrito se e solo se entrambi x_{21} e x_{22} vengono smarriti. Di conseguenza per trovare P_s dobbiamo moltiplicare i P_s dei nodi figli.

$$P_s = 0{,}1 \cdot 0{,}1 = 0{,}01$$

Per poter ottenere lo share x_{11} un attaccante esterno deve riuscire ad ottenere o x_{21} o x_{22}. Pertanto P_c è dato dalla somma dei P_c dei nodi figli.

$$P_c = 0{,}3 + 0{,}3 = 0{,}6$$

Entrambi gli agenti di fatto conoscono x_{11}, quindi per calcolare P_m si devono sommare le P_m dei nodi figli.

$$P_m = 0{,}02 \cdot 0{,}02 = 0{,}04$$

Per quanto riguarda gli attributi di x_{12} si ha $t = 1$ e $n = 2$, i ragionamenti sono analoghi.

$$P_s = 0{,}1 \cdot 0{,}02 = 0{,}002$$

$$P_c = 0{,}3 + 0{,}02 = 0{,}32$$
$$P_m = 0{,}02 + 0{,}005 = 0{,}025$$

Non ci resta che trovare gli attributi di x. Abbiamo $t = 2$ e $n = 2$, il ragionamento è analogo a quello presente nell'esempio precedente.

$$P_s = 0{,}01 + 0{,}002 = 0{,}012$$
$$P_c = 0{,}6 \cdot 0{,}32 = 0{,}192$$
$$P_m = (0{,}04 \cdot 0{,}32) + (0{,}025 \cdot 0{,}6) = 0{,}0278$$

Troviamo infine che

$$R_c = 1 - 0{,}012 = 0.988$$
$$R_s = 1 - 0{,}192 - 0{,}0278 = 0{,}7802$$

In questo caso siamo partiti avendo a disposizione gli stessi agenti del caso precedente, ma utilizzando un albero di distribuzione degli share differente abbiamo ottenuto risultati diversi. Nello specifico, abbiamo migliorato la R_c (0,988 contro 0,68), ma allo stesso tempo abbiamo peggiorato la R_s (0,7802 contro 0,999217).

Albero di distribuzione sbilanciato di profondità 2 Ancora una volta immaginiamo di avere a disposizione quattro agenti. Per tre di questi vale $P_s = 0{,}1$ $P_c = 0{,}3$ $P_m = 0{,}02$ mentre per il quarto abbiamo $P_s = 0{,}02$ $P_c = 0{,}02$ $P_m = 0{,}005$. Questa volta però immaginiamo di utilizzare l'albero di distribuzione degli share sbilanciato di profondità 2 mostrato in figura 5.3.

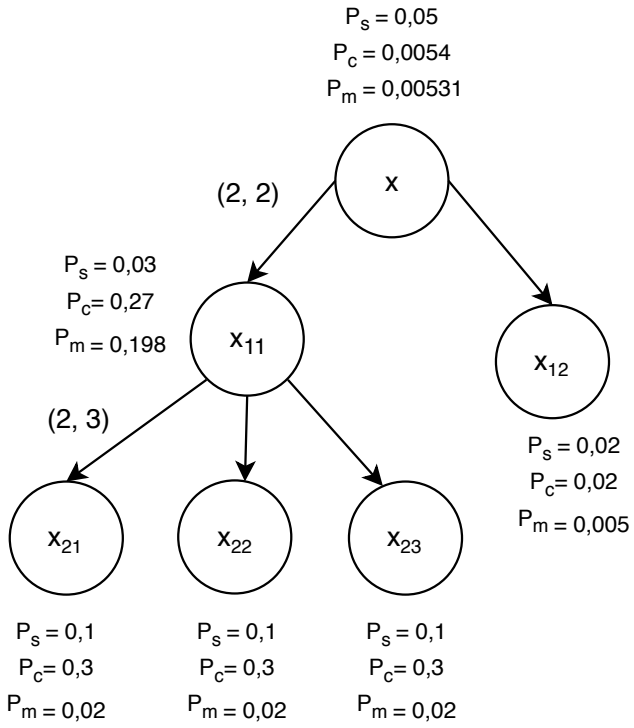


Figura 5.3: Albero di distribuzione sbilanciato di profondità 2.

Per prima cosa cerchiamo gli attributi di x_{11}. Abbiamo $t = 2$ e $n = 3$. La resistenza agli smarrimenti è $\gamma = 1$, perciò P_s è la somma delle probabilità delle tre combinazioni di avere $\gamma + 1 = 2$ smarrimenti.

$$P_s = (0{,}1 \cdot 0{,}1) + (0{,}1 \cdot 0{,}1) + (0{,}1 \cdot 0{,}1) = 0{,}03$$

La resistenza ai furti è $\theta = 2$. Un attaccante deve recuperare due share per ottenere x_{11}. Quindi $P_s = (0{,}3 \cdot 0{,}3) + (0{,}3 \cdot 0{,}3) + (0{,}3 \cdot 0{,}3) = 0{,}27$

Infine P_m è

$$P_m = (0{,}02 \cdot 0{,}3) + (0{,}02 \cdot 0{,}3) + (0{,}02 \cdot 0{,}3) + (0{,}02 \cdot 0{,}3) + (0{,}02 \cdot 0{,}3) + (0{,}02 \cdot 0{,}3) = 0{,}198$$

Non ci resta che trovare gli attributi di x. Ancora una volta $t = 2$ e $n = 2$.

$$P_s = 0{,}03 + 0{,}02 = 0{,}05$$

$$P_c = 0{,}27 \cdot 0{,}02 = 0{,}0054$$

$$P_m = (0{,}198 \cdot 0{,}02) + (0{,}005 \cdot 0{,}27) = 0{,}00531$$

Troviamo infine che

$$R_c = 1 - 0{,}05 = 0{,}95$$

$$R_s = 1 - 0{,}0054 - 0{,}00531 = 0{,}98929$$

Ancora una volta siamo partiti avendo a disposizione gli stessi agenti dei casi precedenti. Con questa struttura di albero di distribuzione degli share differente abbiamo ottenuto un risultato che è un buon compromesso rispetto alle due soluzioni precedenti. R_c è 0,95 contro gli 0,68 e 0.988 precedenti; R_s è 0,98929 contro gli 0,999217 e 0,7802 precedenti.

In generale, non vi è un albero di distribuzione degli share ottimo; data una certa situazione di agenti disponibili la progettazione dell'albero è il threshold su cui possiamo agire per la determinazione delle performance del sistema.

5.3 Considerazioni

Questo modello può essere usato come un comodo strumento in grado di fornire un'indicazione quantitativa sulla robustezza di una data implementazione. Si tratta però di una approssimazione, che non tiene conto di alcune dinamiche che si possono venire a creare. Infatti per poter applicare questa tecnica di analisi si deve assumere che tutti gli agenti siano entità ben distinte e che non vi siano coalizioni tra agenti. Inoltre è necessario attribuire a priori P_s, P_c e P_m ad ogni agente. Nella pratica questa è una attività non banale.

Capitolo 6

Scelta dei parametri del protocollo

In questo capitolo vedremo quali sono i parametri del protocollo e analizzeremo alcuni criteri per il dimensionamento.

6.1 I Parametri

6.1.1 La deadline τ

La deadline τ rappresenta l'istante di tempo in cui si desidera che il segreto venga reso pubblico. Si tratta di un parametro deciso a priori, sul quale difficilmente si riesce ad intervenire. Ad ogni modo è bene sapere che più τ è lontano, più aumentano le possibilità che qualcosa vada storto. La ragione è che più è lontana la deadline, più crescono la probabilità di smarrimento del segreto P_s e l'appetibilità del messaggio da parte di eventuali attaccanti (quindi P_c e P_m).

6.1.2 La tolleranza δ

La tolleranza δ rappresenta quanto ritardo rispetto a τ è accettato per la pubblicazione del segreto. Una tolleranza troppo piccola potrebbe causare problemi se il

client non è abbastanza reattivo al tempo τ, ad esempio per un intasamento del network o perché è temporaneamente offline. Allo stesso tempo una tolleranza troppo grande potrebbe causare un ritardo non accettabile rispetto alla deadline. Per la scelta di questo parametro è ben tenere in considerazione anche la velocità di elaborazione delle transazioni della blockchain utilizzata.

6.1.3 Il *prize*

Il *prize* è la ricompensa da corrispondere all'agente per il suo servizio di detenzione del segreto. È una cifra che deve essere concordata tra le parti. È bene che sia proporzionale a quanto dista τ dal momento dell'inizializzazione del protocollo e a quanto è importante il segreto.

$$prize \propto \tau - t_{init}$$

6.1.4 Il *counterprize*

Il *counterprize* è la cifra che viene corrisposta a chi invia il segreto allo smart contract prima del tempo τ. È bene che

$$prize \gg counterprize$$

Il motivo di questo vincolo è che l'agente potrebbe ottenere il *counterprize* in un qualsiasi istante prima di τ (perché chiaramente conosce lo share) e quindi rendere noto il segreto prima della deadline. Se però il *prize* è molto più grande del *counterprize* sarebbe contro il suo interesse fare questa azione, poiché perderebbe la possibilità di ottenere un guadagno significativamente maggiore al tempo τ.

6.1.5 Il *pawn*

Il *pawn* è la cifra che viene restituita al client quando il segreto viene pubblicato nella finestra temporale $\tau \leq t \leq \tau + \delta$.

Il *pawn* è necessario perché, ovviamente, anche il client conosce il segreto e quindi potrebbe ottenere il *counterprize* (magari poco prima della deadline). Se lo facesse l'agente non potrebbe più ottenere il *prize* anche se si è comportato in maniera corretta, venendo in un certo senso truffato dal client. Come nel caso del *counterprize* si vuole scoraggiare questo comportamento con il vincolo

$$pawn \gg counterprize$$

6.2 Criteri di dimensionamento

6.2.1 Denaro necessario

Il denaro necessario d rappresenta quanto denaro è necessario per inizializzare il protocollo.

$$d = \sum_{i=1}^{N} prize_i + pawn_i + fee_i$$

Nell'ipotesi che tutti i *prize* e tutti i *pawn* siano uguali tra loro e che le commissioni *fee* siano trascurabili l'espressione diventa

$$d = N(prize + pawn)$$

N.B. Questo valore non è il costo che deve sostenere il client, perché parte di questi soldi gli verranno restituiti attraverso i *pawn*.

6.2.2 Costo

Il costo c rappresenta quanto denaro dovrà spendere il client per il servizio richiesto. Il costo non è fissato a priori, perché dipende da quanti *pawn* verranno restituiti.

$$c = \sum_{i=1}^{N} (prize_i + fee_i) + \sum pawn_j$$

dove $\sum pawn_j$ è la somma dei *pawn* restituiti.

Nel caso migliore, ossia quando tutti i *pawn* ritornano al client diventa

$$c = \sum_{i=1}^{N} (prize_i + fee_i)$$

Mentre nel caso peggiore, ossia quando nessun *pawn* ritorna al client, è pari a

$$c = d = \sum_{i=1}^{N} (prize_i + pawn_i + fee_i)$$

Capitolo 7

Applicazioni della Timed-Release Encryption

In questo capitolo presenteremo un elenco non esaustivo di possibili applicazioni della Timed-Release Encryption.

7.1 Offerte in aste a buste chiuse

Spesso per l'assegnazione di bandi pubblici si ricorre all'uso di offerte a buste chiuse. Il funzionamento è il seguente. Vengono fissati i requisiti minimi che devono essere soddisfatti dalle offerte, i criteri per l'attribuzione del punteggio alle offerte e una scadenza. A questo punto ogni partecipante presenta all'ente pubblico la propria offerta in forma cartacea all'interno di una busta sigillata, rimanendo completamente all'oscuro sulla presenza di eventuali altri partecipanti e sul contenuto delle eventuali offerte presentate. Una volta raggiunta la scadenza le buste vengono aperte, validate, vengono calcolati i punteggi e viene decretato il vincitore.

I funzionari pubblici coinvolti hanno un ruolo chiave nel processo. Sono loro a garantire la segretezza e l'integrità delle offerte fino alla scadenza del bando. Inoltre

devono garantire il rispetto della scadenza per la presentazione delle offerte. In questo processo talvolta si verificano casi di corruzione.

Una possibile alternativa è l'uso della Timed-Release Encryption. Ogni partecipante cripta la propria offerta con la TRE, fissando come deadline la scadenza del bando. Raggiunta la scadenza le offerte vengono automaticamente rese pubbliche. A differenza del processo di cui sopra, con la TRE l'integrità e la segretezza delle offerte vengono garantite da proprietà crittografiche e non dall'uomo.

7.2 Insider stock trade

Un insider, ossia una persona che può avere accesso in anticipo a informazioni riservate sull'attività economica di una società quotata in borsa, potrebbe essere legalmente obbligato a prendere impegni in anticipo sull'acquisto di quote della società sia per mitigare potenziali abusi di informazioni interne, sia per proteggere l'insider da false accuse di utilizzo sleale delle informazioni. In certe circostanze si può volere che questi impegni rimangano segreti sino a poco prima della loro esecuzione. Chiaramente un impegno che non garantisce la non ripudiabilità non è sufficiente in quanto un insider può creare in anticipo un impegno nascosto e poi rifiutarsi di renderlo pubblico nel caso in cui l'evento non sia per lui conveniente.

Se un insider cifra in anticipo la sua transazione con la Timed-Release Encryption può sempre essere legalmente obbligato a eseguire la transazione e allo stesso tempo i dettagli della transazione rimangono segreti fino al tempo desiderato. In [38] Rabin e Thorpe propongono un protocollo nel quale gli insider cifrano le loro transazioni in anticipo usando la Timed-Release Encryption. Queste direttive possono essere per l'acquisto, per la vendita o non fare nulla. In questo modo l'insider non rivela informazioni al mercato.

7.3 Raccolta dati in trial clinici

La Timed-Release Encryption può essere utile per garantire l'integrità dei trial clinici. Poiché molti di questi studi sono finanziati da società che sono in grado di guadagnare o perdere ingenti somme di denaro a seconda del loro esito, potenzialmente vi possono essere pressioni sul gruppo di ricerca per ottenere un risultato positivo.

Il protocollo proposto ha due caratteristiche utili in questo caso d'uso. La *proof of existence in time*, ossia è in grado di dimostrare l'esistenza del messaggio al momento dell'inizializzazione; la *non ripudiabilità*, ossia non è possibile in alcun modo impedire il rilascio del messaggio dopo l'inizializzazione ed è sempre dimostrabile chi sia l'autore del messaggio.

L'idea è quella di chiedere ai ricercatori di cifrare i dati con il protocollo proposto man mano che questi vengono raccolti, fissando come deadline la data di termine della raccolta dati. In questo modo raggiunta la deadline i dati diventano disponibili alla commissione di valutazione. L'assunzione è che i dati raccolti acquisiscono significato solo quando raggiungono un certo volume, ossia solo quando ci si avvicina al termine della raccolta dati. Un ricercatore disonesto nelle ultime fasi della raccolta dati potrebbe cercare di manomettere lo studio, generando dati pre-datati o omettendone alcuni. Se però ha cifrato i dati con il protocollo proposto non può farlo, a causa delle due caratteristiche di cui sopra. Cifrando i dati raccolti con il protocollo proposto i ricercatori onesti si tutelano da false accuse e allo stesso tempo i ricercatori disonesti sono impossibilitati dal manomettere a posteriori i dati raccolti.

7.4 Voto elettronico

In alcune forme di votazioni si desidera che non vi siano pubblicazioni di risultati intermedi, perché potrebbero influenzare gli altri elettori. Se i voti vengono cifrati con la Timed-Release Encryption durante la votazione, i risultati possono rimanere interamente segreti fino alla chiusura delle urne.

Capitolo 8

Conclusioni

In questa tesi abbiamo proposto un protocollo per la Timed-Release Encryption. A differenza delle soluzioni precedenti [13, 6, 12, 8, 33, 18, 7], il protocollo proposto è profondamente decentralizzato. Il messaggio è distribuito tra N agenti orchestrati da smart contract eseguiti su blockchain. Nella gran parte delle situazioni il protocollo proposto è in grado di fornire un buon grado di robustezza, ossia è in grado di resistere ad un certo numero di comportamenti scorretti da parte degli agenti senza compromettere la *certezza di pubblicazione* e la *segretezza del messaggio*.

I possibili sviluppi futuri sono diversi. È possibile valutare l'introduzione di uno strumento che utilizza la *zero knowledge proof* per permettere ai destinatari del messaggio di verificare che gli agenti non abbiano smarrito gli share senza dover attendere la deadline. Si può studiare come aggiungere ulteriori disincentivi per dissuadere gli agenti da comportamento scorretti, ad esempio introducendo delle sanzioni attraverso contratti giuridici, appoggiandosi a ledger *permissioned* e meccanismi KYC[1] per validare gli agenti. Si può valutare come gestire messaggi dalle elevate dimensioni che non possono essere economicamente memorizzati nella blockchain

[1] know your customer

appoggiandosi a prodotti commerciali per lo storage in cloud come Amazon S3 [2] e Google Cloud Storage [16] oppure utilizzando storage distribuiti come IPFS [21] e Ethereum Swarm [15]. Si può pensare di creare un marketplace per favorire l'incontro tra domanda (client) e offerta (agenti), incrementando la competizione tra gli agenti con l'obiettivo di ottenere un servizio migliore. In particolare, si può pensare ad un sistema di ranking degli agenti o estendere la variante *cauzione per gli agenti* [si veda 3.3.5] basandosi sull'idea di *proof of stake*. Infine, si può valutare come implementare il protocollo in particolari contesti, magari eliminando la blockchain e utilizzando incentivi/disincentivi non legati alla criptomoneta ma al particolare dominio applicativo.

Bibliografia

[1] ABN Amro Economisch Bureau. *Which problem could the use of blockchain potentially solve for you within the retail chain?* [Accessed October 28, 2018.] 2017. URL: https://www.statista.com/statistics/792463/potential-blockchain-applications-in-retail-in-the-netherlands/.

[2] *Amazon S3*. URL: https://aws.amazon.com/it/s3/.

[3] M. Bellare e S. Goldwasser. *Encapsulated Key Escrow*. Rapp. tecn. Cambridge, MA, USA, 1996.

[4] Ian F Blake e Aldar C-F Chan. «Scalable, Server-Passive, User-Anonymous Timed Release Public Key Encryption from Bilinear Pairing.» In: *IACR Cryptology ePrint Archive* 2004 (2004), p. 211.

[5] blockchain.info. *Number of Blockchain wallet users worldwide from 1st quarter 2015 to 3rd quarter 2018*. [Accessed October 28, 2018.] 2018. URL: https://www.statista.com/statistics/647374/worldwide-blockchain-wallet-users/.

[6] Julien Cathalo, Benoit Libert e Jean-Jacques Quisquater. «Efficient and Non-interactive Timed-Release Encryption». In: *Information and Communications Security*. A cura di Sihan Qing et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 291–303. ISBN: 978-3-540-32099-9.

[7] Konstantinos Chalkias, Dimitrios Hristu-Varsakelis e George Stephanides. «Improved anonymous timed-release encryption». In: *European Symposium on Research in Computer Security*. Springer. 2007, pp. 311–326.

[8] Jung Hee Cheon et al. «Timed-Release and Key-Insulated Public Key Encryption». In: *Financial Cryptography and Data Security*. A cura di Giovanni Di Crescenzo e Avi Rubin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 191–205. ISBN: 978-3-540-46256-9.

[9] Christopher D. Clack, Vikram A. Bakshi e Lee Braine. «Smart Contract Templates: foundations, design landscape and research directions». In: *CoRR* abs/1608.00771 (2016).

[10] *CommonAccord*. URL: `http://www.commonaccord.org/` (visitato il 27/11/2018).

[11] Deloitte. *Blockchain models deployed in organizations worldwide as of April 2018*. [Accessed October 28, 2018.] 2018. URL: `https://www.statista.com/statistics/878706/worldwide-blockchain-models-in-enterprise/`.

[12] Giovanni Di Crescenzo, Rafail Ostrovsky e Sivaramakrishnan Rajagopalan. «Conditional Oblivious Transfer and Timed-Release Encryption». In: *Advances in Cryptology — EUROCRYPT '99*. A cura di Jacques Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 74–89. ISBN: 978-3-540-48910-8.

[13] Yevgeniy Dodis e Dae Hyun Yum. «Time Capsule Signature». In: *Financial Cryptography and Data Security*. A cura di Andrew S. Patrick e Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 57–71. ISBN: 978-3-540-31680-0.

[14] eft Supply Chain & Logistics Business Intelligence. *How is your organization spending on blockchain?* [Accessed October 28, 2018.] 2018. URL: `https://www.statista.com/statistics/827408/spending-blockchain-supply-chain/`.

[15] *Ethereum Swarm*. URL: https://swarm-guide.readthedocs.io/.

[16] *Google Cloud Storage*. URL: https://cloud.google.com/storage/.

[17] Stuart Haber e W Scott Stornetta. «How to time-stamp a digital document». In: *Conference on the Theory and Application of Cryptography*. Springer. 1990, pp. 437–455.

[18] Jung Hee Cheon et al. «Provably Secure Timed-Release Public Key Encryption». In: *ACM Trans. Inf. Syst. Secur.* 11 (mag. 2008). DOI: 10.1145/1330332.1330336.

[19] Stan Higgins. *From $900 to $20,000: Bitcoin's Historic 2017 Price Run Revisited*. 2017. URL: https://www.coindesk.com/900-20000-bitcoins-historic-2017-price-run-revisited (visitato il 14/11/2018).

[20] Stan Higgins. *The Ricardian Contract*. 1996. URL: http://iang.org/papers/ricardian_contract.html (visitato il 27/11/2018).

[21] *IPFS*. URL: https://ipfs.io/.

[22] J. Stark. *Making sense of blockchain smart contracts*. 2017. URL: http://www.coindesk.com/making-sense-smart-contracts/.

[23] Leslie Lamport. «The part-time parliament». In: *ACM Transactions on Computer Systems (TOCS)* 16.2 (1998), pp. 133–169.

[24] Leslie Lamport. «Time, clocks, and the ordering of events in a distributed system». In: *Communications of the ACM* 21.7 (1978), pp. 558–565.

[25] *Legalese*. URL: https://legalese.com/ (visitato il 27/11/2018).

[26] *MAIAN*. URL: https://github.com/MAIAN-tool/MAIAN.

[27] T. C. May. «Timed-release crypto». In: *Manuscript* (1993).

[28] David Mazieres. «The stellar consensus protocol: A federated model for internet-level consensus». In: *Stellar Development Foundation* (2015).

[29] Ralph C. Merkle. «Secure Communications over Insecure Channels». In: *Commun. ACM* 21.4 (apr. 1978), pp. 294–299. ISSN: 0001-0782. DOI: 10.1145/359460.359473. URL: http://doi.acm.org/10.1145/359460.359473.

[30] Satoshi Nakamoto. «Bitcoin: A peer-to-peer electronic cash system». In: (2008).

[31] Arvind Narayanan et al. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[32] Ivica Nikolic et al. «Finding The Greedy, Prodigal, and Suicidal Contracts at Scale». In: *CoRR* abs/1802.06038 (2018).

[33] Kenneth G. Paterson e Elizabeth A. Quaglia. «Time-Specific Encryption». In: *Security and Cryptography for Networks*. A cura di Juan A. Garay e Roberto De Prisco. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–16. ISBN: 978-3-642-15317-4.

[34] PwC. *Biggest barriers for blockchain technology adoption worldwide as of 2018*. [Accessed October 28, 2018.] 2018. URL: https://www.statista.com/statistics/920785/worldwide-blockchain-technology-adoption-barriers/.

[35] PwC. *Stages of blockchain incorporation in the companies in the United States in 2018*. [Accessed October 28, 2018.] 2018. URL: https://www.statista.com/statistics/914259/business-integration-blockchain-stages-usa/.

[36] PwC. *Territories seen as leaders in blockchain technology development worldwide in 2018 and from 2021 to 2023*. [Accessed October 28, 2018.] 2018. URL: https://www.statista.com/statistics/920776/worldwide-blockchain-technology-development-leading-territories/.

[37] PwC. *What business use cases do you most likely see blockchain technology useful for?* [Accessed October 28, 2018.] 2018. URL: https://www.statista.

com/statistics/806306/usage-of-blockchain-technology-among-businesses-in-italy/.

[38] Michael O Rabin e Christopher Thorpe. «Time-lapse cryptography». In: (2006).

[39] Ronald L. Rivest, Adi Shamir e David A. Wagner. *Time-lock puzzles and timed-release crypto*. Rapp. tecn. 1996.

[40] *SECURIFY*. URL: https://securify.chainsecurity.com/.

[41] Adi Shamir. «How to Share a Secret». In: *Commun. ACM* 22.11 (nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.

[42] *Smart Contract Weakness Classification Registry*. URL: https://smartcontractsecurity.github.io/SWC-registry/.

[43] Statista. *Worldwide spending on blockchain solutions from 2016 to 2022, by region (in billion U.S. dollars)*. [Accessed October 28, 2018.] July 2018. URL: https://www.statista.com/statistics/800561/worldwide-blockchain-solutions-spending-by-region/.

[44] *Stellar Smart Contracts Documentation*. URL: https://www.stellar.org/developers/guides/walkthroughs/stellar-smart-contracts.html/.

[45] Nick Szabo. «Formalizing and Securing Relationships on Public Networks». In: *First Monday* 2.9 (1997). ISSN: 13960466. DOI: 10.5210/fm.v2i9.548. URL: http://ojphi.org/ojs/index.php/fm/article/view/548.

[46] Petar Tsankov et al. «Securify: Practical Security Analysis of Smart Contracts». In: *CoRR* abs/1806.01143 (2018).

[47] Dylan Yaga et al. «Blockchain technology overview». In: *Draft NISTIR* 8202 (2018).

Ringraziamenti

Desidero innanzitutto ringraziare il professor Stefano Paraboschi per aver accettato l'incarico di relatore per la mia tesi. Inoltre, ringrazio sentitamente l'ing. Enrico Bacis, l'ing. Dario Facchinetti e l'ing. Marco Rosa per il tempo dedicatomi e per la disponibilità dimostrata nel redimere i miei dubbi durante la stesura del lavoro. Infine, ho desiderio di ringraziare con affetto i miei genitori ed i miei amici per il sostegno dato e per essermi stati vicino ogni momento durante tutto il percorso universitario.