Universitat de Lleida

TREBALL FINAL DE GRAU

ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: **Arnau Díaz Torres**

Titulació: Grau en Enginyeria Informàtica

Títol de Treball Final de Grau:

Disseny d'un sistema de votació en Blockchain.

Director/a:

Josep Argelich Roma

Josep Maria Miret Biosca

Presentació

Mes: Septembre

Any: 2018

Índex

Índex de figures

Capítol 1

Introducció

Introduirem el treball parlant de les nostres motivacions, els objectius que ens hem plantejat durant la realització del mateix i un petit resum de l'estructura de la memòria.

1.1 Motivació

Ens els últims anys s'han viscut grans avanços en camps de la informàtica com la intel·ligència artificial, tractament i anàlisi de dades, criptografia, seguretat... Camps on la recerca havia trobat protocols i algoritmes que a causa de les limitacions físiques no s'havien pogut implementar en el passat però en els últims anys gràcies al paral·lelisme de les tasques i la millora en el hardware s'estan veient salts importants no sols en el camp teòric sinó en aplicacions que ens aporten beneficis socials i empresarials en la realitat.

Una idea que va sorgir a principis de segle XX juntament amb la popularitat que començava a tenir internet i en concret el seu comerç, va ser la idea de desenvolupar un protocol per a realitzar transaccions sense passar per institucions financeres que seguissin intermediàries en les operacions [2]. Les signatures digitals van ser un avanç cap a l'objectiu però els beneficis que suposaven sobretot en forma d'anonimització es veien contrarestats per la dificultat de controlar el $double - spending$, com evitar que una persona utilitzés dos cops la mateixa moneda en diferents transaccions. La proposta de solució a aquest problema que s'ha agafat amb més consideració és la de tenir una xarxa $peer - to - peer$. La xarxa conté *timestamps* de les transaccions que les emmagatzema mitjançant *hashes* en una cadena que utilitza proves de treball per assegurar la seva robustesa. D'aquesta manera Satoshi Nakamoto va proposar *Bitcoin* [5] en 2009.

La *blockchain* proposada i els algoritmes que utilitzava eren ja coneguts i s'havien estudiat en el món acadèmic amb anterioritat, però la majoria fins a l'aparició d'aquests sistemes descentralitzats i les seves característiques concretes

mai havien tingut un propòsit real i que fos funcional a escala d'implementació. L'evolució de les cadenes de blocs i els sistemes descentralitzats on garantim la seguretat de les transaccions han evolucionat en els últims anys a l'aparició de les anomenades noves generacions de *blockchain*. Les quals ja no només estan fetes per a donar suport a transaccions que representen monedes sinó que permeten registrar propietat i prendre decisions dins de la xarxa de forma autònoma. Van néixer així la generació 2.0 i 3.0 [16] de cadena de blocs, que mitjançant codi que viu en la xarxa que s'executa de forma segura en tots els nodes, es pot interactuar com es desitja amb ella, per exemple fer una votació, registrar propietats, passar la possessió de propietats entre usuaris...

Nosaltres en aquest treball explorarem l'estat de l'art de la tecnologia i discutirem els beneficis i inconvenients en crear protocols o sistemes criptogràfics que s'aprofitin de les característiques de la cadena de blocs.

1.2 Objectius

Dins d'aquest treball ens hem proposat dissenyar un sistema criptogràfic o protocol de votació electrònica en *blockchain*. A aquest document l'acompanya una implementació que verifica les idees que es proposen i la viabilitat de realitzar el projecte, no és un prototip funcional al complet però s'han implementat tests per assegurar la fiabilitat tan de la criptografia com de la interacció amb la cadena de blocs.

Per tal de dissenyar aquest protocol i com discutirem en els capítols preliminars, s'han de tenir clars els requisits de seguretat i el tipus de votació que es vol realitzar. En el nostre cas es va proposar realitzar una votació on donat un grup de membres i una pregunta, es poden votar diferents respostes i un cop tothom ha votat es pot obtenir un resultat. A continuació anem a detallar en més profunditat l'objectiu del sistema i en futurs capítols veurem detalls de la implementació i diferents alternatives als apartats.

- Tenim un grup de persones V, que representa als votants.
- Tenim a una autoritat A, que pot ser membre de V o no.
- La votació té tres fases, confirmació, votació, recompte.
  - En la fase de confirmació, totes les persones confirmaran a la autoritat A la voluntat de votar.
  - En la fase de votació els votants realitzaran el seu vot.
  - En la fase de recompte l'autoritat A donarà el resultat de la votació.

Ara comentarem a mena d'avanç algunes de les opcions de seguretat que existeixen [18][17]:

- Un vot ha de ser correcte per a ser tingut en compte i aquesta prova no dona informació sobre el vot, només ens assegura la correctesa. Podem incloure proves de domini.
- Els votants encripten el seu vot.
- Els votants podrien emmascarar la seva identitat i separar-la del seu vot.
- Els votants podrien confirmar que el seu vot ha sigut correctament comptat.
- Els votants podrien generar vots falsos que es pugessin confirmar com correctes. Per exemple en el cas d'un tercer intentant extorsionar un votant, un votant el podria enganyar i votar el que volgués.
- Els votants poden accedir a l'estat de la votació en tot moment en temps real.
- El recompte es fa de forma anònima per al votant, és a dir, l'autoritat A mai coneix el que ha votat un votant.
- Fins que no han votat tots, l'autoritat A no pot saber el resultat final o serà un resultat invàlid depenent de com definim la criptografia.

Aquest últim punt és important, és una decisió que hem agafat per tal de que el protocol pugui tenir les opcions de seguretat que la votació requereixi. En el nostre cas aquest punt ha influenciat molt la fase de recompte com veurem en l'apartat on ho comentem.

Una altra consideració important a tenir en compte és la limitació de *Ehereum* a l'hora d'executar codi. Aquest treball ha explorat les possibilitats i les relacions que hi ha a nivell de protocol. Si volguéssim implementar algoritmes molt sofisticats per a ser executats en la cadena de blocs, s'hauria de tenir en compte tant la gran complexitat que això suposaria al programador a causa de les limitacions de les estructures de dades a l'hora de programar, com també la complexitat en el temps a l'executar-se.

1.3 Estructura de la memòria

Per tal de poder implementar un sistema criptogràfic en un entorn concret i amb el comportament que hem plantejat, hem d'introduir una sèrie de conceptes que considerem bàsics. Les tres branques principals són criptografia, votació electrònica i blockchain.

Dins d'aquest treball hem considerat per a cada problema que volíem solucionar o per cada element que ha de complir el protocol vàries solucions. En l'apartat preliminar d'aquesta documentació només es comentarà el que creiem més adequat per tal de comprendre la veracitat del protocol que hem acabat

implementat. En les seccions principals del cos del treball quan comentem els algoritmes concrets utilitzats comentem aquestes alternatives que hem estudiat però no es troben explicades en la secció preliminar del treball.

Sobre la votació electrònica s'ha de comentar que existeixen moltíssims algoritmes i tècniques criptogràfiques concretes depenen del tipus de votació que necessitem, però a nosaltres només ens interessa la formalitat del que hem acabat implementant. A causa del gran nombre d'opcions també citarem les característiques bàsiques de caràcter general que s'han de complir en un sistema.

Pel que fa a la part d'implementació tenim dues seccions:

- Disseny, on expliquem el protocol de votació que hem escollit i els avantatges i inconvenients que hem trobat.
- Aplicació, comentem l'entorn de desenvolupament que hem elegit per programar amb Ethereum, la criptografia i l'aplicació web que acompanyen aquest treball.

Capítol 2

Preliminars

Comencem aquest treball amb un exemple [8] senzill de criptografia aplicada a resoldre un problema. Aquest petit exemple ens servirà per veure tres temes principals en la criptografia. Després donem una introducció a la cadena de blocs pel lector que desconegui els conceptes bàsics i expliquem les noves cadenes que permeten executar codi.

2.1 Criptografia

En aquesta secció veurem els següents punts per a seguir el treball en futurs apartats:

- Aportar una demostració inicial de l'efectivitat i la practicitat d'utilitzar criptografia per resoldre els problemes de les aplicacions.
- Suggerir una primera idea de en què es basa la criptografia.
- Començar a establir la mentalitat necessària per desenvolupar un sistema criptogràfic per a la seguretat de la informació.

Llavors comencem amb un joc com a exemple trivial que trobarem que es complica quan els membres del joc estan remotament allunyats entre ells. El problema és que els membres no confien els uns amb els altres. Per tal de tenir un joc just necessitem una mena de protecció.

2.1.1 Problema de comunicació

Aquest és el joc que comentàvem. Alice i Bob són dos amics que volen passar la tarda junts però no es decideixen si anar al cinema o a l'òpera. Però arriben a l'acord de deixar que una moneda decideixi. Cadascú tria una cara de la moneda i si en tirar-la a l'aire surt el seu costat aquesta persona decidirà el lloc on passaran la tarda.

Donat un exemple tan senzill podem observar dins del món de l'estudi de les comunicacions que ens trobem davant d'un protocol. Un protocol és un procediment ben definit que es realitza entre un nombre plural d'entitats participants. És important la pluralitat dels participants, si un procediment només es executat per un participant no podem parlar de protocol.

2.1.2 Aplicant criptografia

Ara imaginem que els dos amics volen dur a terme aquest protocol per telèfon. Alice diu a Bob, "Tria un costat. Llavors tiraré la moneda i et diré si has guanyat o no."Òbviament Bob no estarà d'acord perquè no pot verificar el resultat.

En aquesta situació és on la criptografia entra en joc. Busquem una funció matemàtica $f(x)$ que per cada enter té les propietats següents:

1. Per cada enter x, és computacionalment fàcil calcular $f(x)$, però és computacionalment difícil [12] trobar informació sobre x donada la seva imatge.
2. És impossible trobar dos enters (x, y) que compleixin $x \neq y$ i a la vegada $f(x) = f(y)$.

Donades aquestes propietats podem definir el protocol següent:

Premissa: Alice i Bob han acordat,

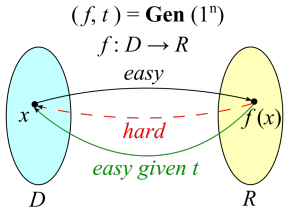
1. Una funció $f(x)$ que compleixi les propietats especificades anteriorment.
2. Un nombre parell x representi cara i un altre imparell que representi creu.
3. Alice agafa un número gran i calcula $f(x)$, llegeix $f(x)$ per telèfon.
4. Bob li diu a Alice si creu que ha sortit parell o imparell.
5. Alice llegeix x a Bob.
6. Bob verifica $f(x)$ i veu si és correcte.

Anem a veure la seguretat del protocol, primer Alice no pot trobar dos nombres que siguin parells i imparells al mateix temps. Llavors un cop a donat la imatge d'aquest valor per telèfon no podrà canviar la seva elecció. Segon, donada la propietat de la funció $f(x)$, Bob no pot determinar quin era el valor inicial.

Aquest protocol utilitza un dels requeriments fonamentals en la criptografia moderna: una funció unidireccional [15]. Les dues propietats de la funció posseeixen dos problemes computacionalment intractables, un per l'Alice i un altre per el Bob.

Llavors podem generalitzar dient que tot sistema criptogràfic implica l'existència d'una funció unidireccional com en la figura següent.

Figura 2.1: Funció unidireccional.



2.1.3 Criteris per a un bon sistema criptogràfic

- **Rigorositat de la protecció**

  Depenent de les nostres necessitats necessitarem un sistema més rigorós o menys. En el petit exemple vist anteriorment no hem definit quina funció utilitzàvem per encriptar el resultat. Aquesta funció pot ser més segura i més difícil de calcular si per exemple és un valor que serà observat per molta gent com per exemple en un sistema en xarxa. Per una altra banda podem tenir sistemes interns que s'aprofiten d'algunes propietats criptogràfiques i en aquests casos no és necessari utilitzar funcions tan complexes.

- **Confiança en la seguretat**

  Com podem assegurar-nos que un sistema o un protocol són segurs? Normalment, la tasca de trencar un sistema criptogràfic va associada a resoldre un problema matemàtic. Aquests problemes són considerats difícils o intractables des del punt de vista computacional, no poden ser resolts en un temps raonable.

  Hi ha un nombre de problemes intractables utilitzats de forma habitual en estàndards en protocols de criptografia moderna. En particular, en sistemes de clau publica o asimètrica. Alguns d'aquests problemes inclouen el problema de la factorització d'enters, el problema del logaritme discret[13], Diffie-Hellman[3] i alguns problemes associats.

- **Eficiència pràctica**

  Un protocol criptogràfic no és només un algoritme, també és un procediment de comunicació que implica transmetre missatges sobre la xarxa entre diferents participants. Llavors un protocol té una nova dimensió per a mesurar l'eficiència: el nombre d'interaccions de comunicació.

Normalment un pas de comunicació és més costos que un pas executat localment (per exemple l'execució d'un càlcul d'unes dades que tenim en una màquina). Si haguéssim de comunicar les dades entre els membres de la xarxa, estaríem afegint un cost elevat al procés. Per tant és desitjable que un protocol criptogràfic tingui el mínim nombre d'interaccions possible.

El criteri d'eficiència estàndard per declarar si un algoritme és: comprovar si el temps d'execució està acotat per un factor polinòmic del problema. Si apliquem aquest mateix criteri a un protocol, un protocol eficient hauria també d'estar acotat per un factor extremadament petit, una constant o com a molt una funció lineal.

Un protocol de comunicació en el que el nombre d'interaccions excedeixi una funció lineal no hauria de ser considerat com eficient i per tant, no hauria de ser un bon protocol en un cas pràctic.

2.1.4 Criptosistema clau pública

La criptografia de clau pública, o criptografia asimètrica, és qualsevol sistema criptogràfic que utilitza parells de claus: claus públiques que es poden difondre i claus privades que només coneix el propietari. D'aquesta manera aconseguim dues funcions:

- **Autentificació**, on la clau pública verifica que un titular de la clau privada ha enviat el missatge.
- **Encriptació**, on el titular de la clau privada pot desxifrar el missatge encriptat amb la clau publica.

En un sistema de xifrat de clau pública, qualsevol persona pot xifrar un missatge utilitzant la clau pública del receptor. Aquest missatge encriptat només pot ser desxifrar amb la clau privada del receptor. La generació d'un parell de claus públiques i privades ha de ser eficient des del punt de vista computacional. La robustesa d'un sistema criptogràfic com hem comentat en la secció anterior depèn de l'esforç computacional necessari per trobar la clau privada a partir de la seva clau publica emparellada. La seguretat depèn llavors de mantenir la clau privada oculta a tercers.

Actualment RSA[7] i ElGamal[4] són els algoritmes de clau pública més utilitzats, en el nostre cas anem a aprofundir en ElGamal.

ElGamal

ElGamal és un algoritme de xifratge de clau pública i la seguretat de l'algoritme es basa en el problema de logaritme discret. Com hem vist amb anterioritat,

resoldre un sistema criptogràfic normalment va associat a resoldre un problema matemàtic amb una complexitat molt alta. El procediment de xifratge està basat en càlculs sobre un grup cíclic. Això determina que la seguretat depengui directament de la dificultat de calcular un logaritme discret sobre aquest grup cíclic.

- **Inicialització del sistema:** En tot sistema criptogràfic de clau pública necessitem generar les claus, tant la pública com la privada per a distribuir-les, l'algoritme genèric seria el següent:

  1. Considerem un grup cíclic multiplicatiu G d'ordre q i calculem un generador g de G.
  2. Escollim una clau privada x de forma aleatòria en un rang $1, ..., q-1$.
  3. Calculem la clau pública $y = g^x$
  4. Treballem en un subgrup d'ordre q de $\mathbf{Z}_p^*$ on escollim $p = 2 \cdot q + 1$ amb p i q primers.

- **Xifratge:**

  1. Elegim de forma aleatòria r entre $1, ..., q-1$ i calculem $c_2 = g^r (mod\ p)$.
  2. Convertim el missatge que volem encriptar en un element m de G.
  3. Calculem $c_1 = m \cdot y^r (mod\ p)$.
  4. El xifrat serà la tupla (c_1, c_2).

- **Desxifratge:** Donat un xifrat $(c1, c2)$ hem de recuperar m.

  1. Calculem $y^r = c_2^x$.
  2. Calculem $m = \dfrac{c_1}{y^r (mod\ p)}$.

  On podem observar la correctesa donat que:

$$m = \frac{c_1}{y^r} = \frac{m \cdot y^r}{y^r} = m$$

ElGamal additiu

Com veurem en la secció de disseny, un dels motius principals pel que hem escollit ElGamal és la facilitat amb la que podem dotar-ho de la propietat de ser un homomorfisme additiu[21]. El qual complirà les propietats següents[19][20]:

- La multiplicació de dos textos xifrats desxifrarà com la suma dels dos textos en clar:

$$D(E(m_1)) \cdot E(m_2) = m_1 + m_2$$

- **Inicialització del sistema:**
  1. Considerem un grup cíclic multiplicatiu G d'ordre q i generador g de G.
  2. Escollim una clau privada x de forma aleatòria en un rang $1, ..., q-1$.
  3. Calculem la clau pública $y = g^x$
  4. Treballem en un subgrup d'ordre q de $\mathbf{Z}_p^*$ on escollim $p = 2 \cdot q + 1$ amb p i q primers.
- **Xifratge:**
  1. Elegir de forma aleatòria r entre $0, ..., q-1$ u calculem $c_2 = g^r (mod\ p)$.
  2. Convertim el missatge que volem encriptar m en un enter k tal que $1 \leq k \leq q-1$.
  3. Calcular $c_1 = g^k \cdot y^r (mod\ p)$.
  4. El xifrat serà la tupla (c_1, c_2).
- **Desxifratge:** Donat un xifrat (c_1, c_2) hem de recuperar m.
  1. Calculem $y^r = c_2^x (mod\ p)$.
  2. Calculem $g^k = \dfrac{c_1}{y^r} (mod\ p)$.
  3. Busquem en una taula pregenerada el valor de k.

2.2 Blockchain

La *blockchain*[5] és una estructura de dades ordenada, que conté una llista enllaçada de blocs de transaccions, també anomenada cadena de blocs. Cada bloc en la blockchain és identificat per un *hash*[14], generat utilitzant algoritmes com $SHA256$ que s'emmagatzema en la capçalera del bloc. Cada bloc a més a més referència el bloc anterior en la cadena, anomenat com el bloc pare, utilitzant el seu *hash* identificatiu. La seqüència de hashes enllaçant-se al seu pare crea una cadena que va fins al primer bloc mai creat, conegut com el bloc gènesis.

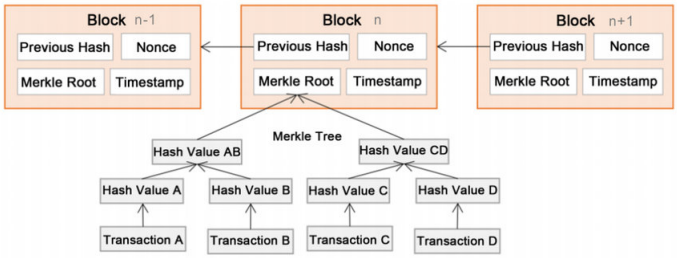


Figura 2.2: Típica estructura d'una cadena de blocs

Tot i que un bloc només té un pare, pot temporalment tenir múltiples fills. Aquesta situació és dona sense entrar en detall mentre es descobreixen nous blocs simultàniament, quan finalment només un fill acaba formant part de la blockchain és continua la cadena amb normalitat.

El camp que conté el hash previ dintre d'un bloc afecta el hash resultant d'aquest. La identitat del fill canviarà depenent de la identitat del pare. Quan un pare és modificat en qualsevol manera, el seu hash canvia. Aquest canvi obliga al seu fill a actualitzar el camp de hash previ, que farà canviar el seu hash propi i així consecutivament. Aquest efecte de cascada assegura que un cop un bloc té moltes generacions, no es pot canviar sense forçar el recàlcul de tots els blocs subseqüents.

Com que aquest càlcul és tan costós es requeriria una capacitat de computació enorme (i per tant un consum energètic equivalent), l'existència d'una cadena de blocs llarga implica que l'historial d'aquesta cadena és immutable, aquesta és una de les claus de la blockchain.

2.2.1 Estructura d'un bloc

Per introduir l'estructura d'un bloc ho veurem sobre un bloc de Bitcoin [1] ja que és més senzill i conté les idees principals. En seccions posteriors introduirem i les diferències més significants.

- **Estructura:** Un bloc és un contenidor de dades que agrega transaccions per a incloure-les en la cadena de blocs. El bloc està constituït d'una capçalera, contenint metadades, seguida d'una llista de transaccions.

Size	Field	Description
4 bytes	Block Size	Size of the bloc
80 bytes	Block Header	Several fields from the header
1-9 bytes	Transaciton Counter	How many transactions follow
Variable	Transactions	The transactions in this block

Taula 2.1: Observem les dades en un bloc de transaccions.

- **Capçalera:** La capçalera consisteix de tres sets de metadades. Primer, hi ha una referència al bloc previ, que connecta aquest bloc amb el bloc pare en la cadena. Segon set, es la $dificulty$, *timestamp* i el *nonce* relacionats amb la mineria dels blocs. El tercer set és el *merkleroot*, una estructura de dades que veurem que s'utilitza per compactar de forma eficient totes les transaccions d'un bloc.

Size	Field	Description
4 bytes	Version	Version number to track protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block
4 bytes	Timestamp	The approximate creation time of this block
4 bytes	Difficulty	The Proof-of-Work algorithm difficulty target
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

Taula 2.2: Observem els camps en una capçalera d'un bloc de Bitcoin.

- **Block header hash:** L'identificador principal d'un bloc és un *hash* criptogràfic, una empremta digital, calculada utilitzant algoritmes de SHA.
- **Merkle Trees:** Cada bloc en la *blockchain* conté un resum compacte de totes les seves transaccions utilitzant un *merkletree*.

  Un *merkletree*, també conegut com *binaryhashtree*, és una estructura de dades utilitzada per la seva eficiència en verificar la integritat d'un set de

dades. Els *merkletrees* són arbres binaris que contenen *hash* criptogràfic. En un bloc de transaccions el que aconseguim utilitzant-los és una empremta digital del set sencer de transaccions d'un bloc, donant-nos una manera molt eficient de comprovar si una transacció està dins d'un bloc.

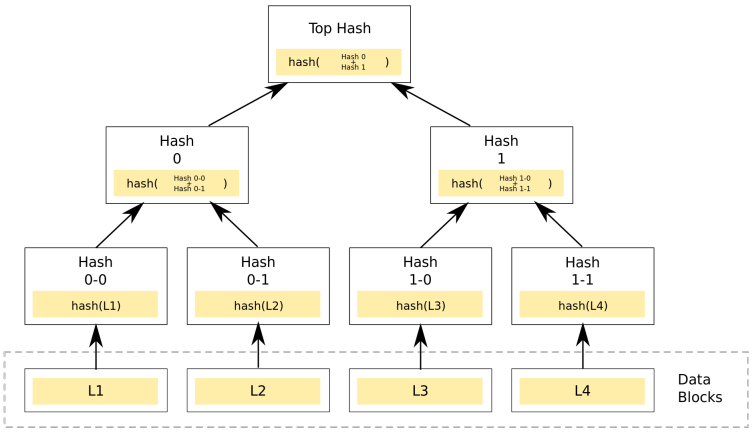


Figura 2.3: Arbre de hash.

Si tenim un *merkletree* amb N elements podem comprovar l'existència de qualsevol element en $2 \cdot \log_2 N$ càlculs.

Number of transactions	Size of the block	Path (Hashes)	Bytes
16 transactions	4 kilobytes	4 hashes	128 bytes
512 transactions	128 kilobytes	9 hashes	288 bytes
2048 transactions	512 kilobytes	11 hashes	352 bytes
65632 transactions	16 megabytes	16 hashes	512 bytes

Taula 2.3: Dades que s'ha d'intercanviar per provar correctesa d'un bloc.

La taula 2.3 mostra la informació que s'ha d'intercanviar en un *path* per a demostrar que una transacció forma part d'un bloc. Com podem observar, mentre que el tamany de bloc incrementa ràpidament, el camí requerit per demostrar la inclusió d'una transacció incrementa molt a poc a poc. Amb un *merkletree*, un node pot descarregar només les capçaleres i ser capaç d'identificar si hi ha una transacció consultant un petit camí, que a la pràctica poden ser gigabytes. Els nodes que no mantenen una cadena de blocs, utilitzen aquest mecanisme per a verificar transaccions de forma eficient.

2.2.2 Ethereum

- **Consens descentralitzat**

  Com hem comentat Bitcoin no només va aportar una solució al problema de tenir una moneda virtual sinó potser més important va ser l'establiment de la tecnologia de cadena de blocs com a eina per a establir un consens distribuït.

  Algunes de les aplicacions alternatives inclouen l'ús d'actius digitals en les cadenes per representar monedes i instruments financers, la propietat d'un dispositiu físic, actius no fungibles com els del nom del domini, així com aplicacions més complexes en el que els actius digitals estan controlats directament per un fragment de codi que implementa regles arbitràries o fins i tot organitzacions autònomes descentralitzades.

  Ethereum proposa una cadena de bloc que pot ser utilitzat a diferència de Bitcoin per a crear contractes intel·ligents dins la xarxa, aquests contractes poden codificar funcions de transició d'un estat a un altre, permeten executar a l'usuari qualsevol d'elles, d'aquesta manera tenim la capacitat de que tots els usuaris comparteixin un codi i puguin interactuar amb ell.

  Abans d'explicar Ethereum anem a comentar les dues formes cap a adquirir un protocol de consens: construir una xarxa independent i construir un protocol per sobre de Bitcoin. La primera proposta, tot i ser viable i funcionar correctament en aplicacions com Namecoin[9], és difícil d'implementar. Cada implementació individual necessita arrancar una cadena de blocs independent, així com construir i provar tots els estats de transacció i el codi de la xarxa necessària. A més a més, la gran majoria d'aplicacions serien massa petites per a garantir la seguretat de la seva pròpia cadena.

  SPV, $Simplified\ Payment\ Verification$, funciona en Bitcoin perquè pot utilitzar la profunditat d'un bloc en la cadena com a validesa. Els protocols basats en cadena de blocs, no poden fer que la cadena de blocs no inclogui transaccions que no son vàlides dintre del context del seu propi protocol. Per tant una implementació SPV completament segura necessitaria escanejar tota la cadena de blocs de Bitcoin per detectar si certes transaccions són vàlides o no.

  En l'actualitat, totes les implementacions de cadenes de blocs que estan basats en Bitcoin, es basen en la confiança en un servidor de confiança per proporcionar dades, el que no és una solució òptima, ja que, precisament un dels objectius principals de les criptoconcurrencies és eliminar la necessitat d'aquest servidor de confiança.

- **Característiques**

  La intenció d'Ethereum s'apropa molt als nostres objectius, Ethereum és un protocol alternatiu per a la construcció d'aplicacions descentralitzades amb especial èmfasi en: desenvolupament ràpid, seguretat en aplicacions petites i poc utilitzades i la capacitat de les diferents aplicacions d'interactuar de forma molt eficient.

  Ethereum ho aconsegueix permetent l'execució de contractes intel·ligents dins del seu protocol, on qualsevol usuari pot crear aplicacions descentralitzades amb les seves regles arbitràries per a la propietat, formats de transacció i funcions de transició d'estat. Una versió bàsica de Namecoin[9] es pot escriure en dues línies de codi i altres protocols com divises i sistemes de reputació es poden aconseguir en menys de vint línies.

- **Comptes**

  En Ethereum, l'estat es compon d'objectes anomenats çomptes", cada una de les quals té una direcció i les transaccions d'estat són transferències directes de valor i informació entre contes. Un compte conté els camps següents:

  – El *nonce*, un comptador utilitzat per la $proof - of - woork$.
  – El balanç de *ether* del compte.
  – El codi de contracte e si està present.
  – L'emmagatzematge del compte, buit per defecte.

  El *ether* és el principal çriptocombustibleïntern d'Ethereum i s'utilitza per pagar les tasses de transacció, el qual ens podem referir com tx. En general hi ha dos tipus de comptes: les controlades per claus privades i les comptes de contracte, controlades pel codi del contracte.

  Un compte de propietat extern no té codi i es poden enviar missatges des d'un compte en propietat extern mitjançant la creació d'una firma de transacció. En un compte de contracte, cada cop que el compte rep un missatge s'activa el seu codi, el que permet llegir i escriure l'emmagatzematge intern i enviar altres missatges a altres comptes. Aquests contractes viuen dins de l'entorn d'execució i sempre s'executen de la mateixa manera. És a dir, tots els nodes de la xarxa executen el mateix codi.

- **Missatges i transaccions**

  És important a l'hora de construir un protocol de votació tenir en consideració quina informació forma part d'un missatge i quina queda emmagatzemada en la cadena de blocs, si en el futur es trenqués l'encriptació com

ens afectaria? El terme transacció s'utilitza per a referir-nos als paquets de dades firmades que emmagatzema un missatge per ser enviat des d'un compta de propietat externa. Contenen:

- Destinatari del missatge.
- Una signatura digital identificant el compte que envia.
- La quantitat de *ether* a transferir.
- Un camp opcional de dades.
- Un valor $STARTGAS$, representa el màxim de passos computacionals que l'execució de la transacció pot realitzar.
- Un valor $GASPRICE$, representa la tarifa que el remitent paga per cada pas computacional.

Anem a comentar els dos últims camps, ja que són molt rellevants en qualsevol aplicació descentralitzada. Aquests camps són crucials per al model capaç d'evitar atacs de negació de servei. Per evitar bucles infinits accidentals o atacants o altres errors computacionals en el codi, es requereix que cada transacció tx estableixi un límit de quants passos d'execució de codi pot utilitzar. La unitat fonamental és el *gas*, normalment un pas computacional costa 1 gas, però altres operacions costen més o augmenten la quantitat de dades que han d'emmagatzemar-se com a part de l'estat. D'aquesta manera podem calcular *benchmarks* per analitzar els costos d'executar el codi i poder analitzar la seva viabilitat en casos reals.

Els contractes tenen la capacitat d'enviar missatges a altres contractes. Els missatges són objectes virtuals que mai s'inverteix una serialització i per tan només existeixen en l'entorn d'execució d'Ethereum. Un missatge està format per:

- El remitent del missatge.
- El destinatari.
- La quantitat d'ether a transferir.
- Un camp de dades opcional, on per exemple podem incloure les dades que volem que rebi un contracte.
- Un valor $STARTGAS$.

Essencialment un missatge és com una transacció, excepte que és produït per un contracte. Es produeix un missatge quan un contracte està executant codi i executa l'operand $CALL$, que produeixi i executa el missatge. Per tant els contractes es poden comunicar entre ells. En el nostre cas de sistema criptogràfic és rellevant aquesta distinció perquè si un contracte només pot ser cridat per comptes d'usuari no serà atacat per contractes tan fàcilment com si no ens preocupéssim.

- **Execució de codi**

  El codi en els contractes Ethereum està escrit en un llenguatge bytecode de baix nivell anomenat "EVM code". El codi consisteix en una sèrie de bytes, on cada byte representa una operació. Pel que fa al desenvolupament d'aquest treball no entrarem gaire en detall en les capacitats o en el funcionament formal d'aquest. Les operacions tenen accés a tres espais on emmagatzemen dades:

  – La pila.
  – Memòria, una matriu de bytes infinitament ampliable.
  – L'emmagatzemament a llarg termini del contracte, un emmagatzemant clau-valor. A diferència de la pila i la memòria, és ajustat un cop finalitzat el càlcul.

Capítol 3

Votació electrònica

En aquest capítol de la memòria introduirem els conceptes bàsics d'una votació tradicional i després els relacionarem amb el seu significat en un entorn de cadena de blocs, quins problemes hi pot haver i quines possibles representacions existeixen.

3.1 Votació electrònica tradicional

Parlarem de les fases en què dividim el procés de votació tradicional, cadascuna amb uns processos interns que pretenen simular els processos d'una votació tradicional. Aquestes fases són:

- **Fase de preparació.**
    1. **Anunci de l'elecció:** L'autoritat de l'elecció publica el propòsit de l'elecció, la llista de candidats, requisits, condicions, etc.
    2. **Registre de votants:** Es busca aconseguir informació sobre els votants per a la seva posterior identificació.
- **Fase de votació.**
    1. **Autenticació del votant:** El votant s'autentica per poder emetre el vot.
    2. **Establiment de sessió:** Es relaciona el votant amb el servidor de vot al qual es connecta per efectuar la votació.
    3. **Selecció de candidats:** Es mostra la llista de candidats per tal que el votant esculli.
    4. **Enviament del vot:** Un cop escollit el candidat, es xifra l'elecció del votant i s'envia al servidor corresponent per a la seva tramitació.
    5. **Validació del vot:** Es verifica que el votant només ha votat un cop i que el vot emès és correcte.

- **Recompte i publicació.**

    1. **Transferència de vots:** Quan ha acabat la fase de votació l'autoritat de votació transfereix els vots a l'autoritat d'escrutini.
    2. **Permutació de vots:** S'utilitza una funció de permutació de vots per trencar qualsevol lligam entre el votant i el seu vot. S'acostuma a combinar amb un re xifratge.
    3. **Desxifratge de vots:** Es recupera el vot original per al seu posterior re- compte.
    4. **Escrutini.**
    5. **Auditoria de resultats:** Comprovació que els resultats són correctes i que no hi ha hagut cap manipulació tant en la recol·lecció com en el recompte.
    6. **Publicació de resultats.**

3.1.1 Requisits de seguretat

La principal sospita que recau sobre un procés electrònic és la seva seguretat d'avant d'un procés que es realitza manualment. Per tant qualsevol possibilitat d'intervenció no desitjada en el procés ha de ser controlada, tant si prové de fora com de dintre del propi sistema.
Els següents punts constitueixen una descripció dels requisits de seguretat que ha de complir una votació electrònica:

- **Legitimitat del votant:** Només poden participar votants autoritzats i només es tindrà en compte un vot per votant. Normalment s'utilitza un cens per identificar els votants autoritzats. En el cas de la votació electrònica s'utilitzen tècniques d'identificació remota.

- **Robustesa:** Un votant només pot realitzar un vot i ningú fora del cens pot efectuar una votació.

- **Confidencialitat:** La relació entre el votant i el seu vot no pot ser coneguda ni deduïble.

- **Precisió:** El resultat de la votació ha de venir donat pel recompte de tots els vots recollits de manera legítima. S'ha d'evitar qualsevol alteració dels vots mitjançant sistemes de prevenció i detecció d'alteracions.

- **No informació:** No es poden conèixer resultats parcials durant la votació, ja que el coneixement de l'estat de la votació podria influir en els votants que encara no han votat.

- **Verificació:** Pot ser de dos tipus:

    - **Individual:** El votant ha de poder comprovar que el seu vot ha estat enviat, rebut i processat correctament.

  - **Universal:** És important que hi hagi una comprovació pública per a que qualsevol participant i/o observador pugui verificar la integritat del resultat.

- **Evitar coercions:** Un votant no hauria de poder demostrar a un tercer el vot que ha escollit, ja que una tercera persona podria exigir al votant que votés una opció en concret. Si no ho pot demostrar aquesta tercera persona no pot saber si ha votat el que ell ha exigit o no.

3.2 Votació en blockchain

La presa de decisions col·lectiva en general, i el vot en particular, representen una proposta de valor principal per a la cadena de blocs Ethereum. Molts projectes depenen de mecanismes de votació acuradament calibrats. Aquesta secció discuteix alguns dels desafiaments del mecanisme de votació d'Ethereum, i com es poden fer justos i segurs.

- **Com representem un vot**

  A diferència del protocol de Bitcoin, el protocol d'Ethereum va ser dissenyat explícitament per fer més que simplement crear i registrar transferències de les seves pròpies criptomonedes. És un protocol més generalitzat i permet crear altres *tokens* en la seva blockchain.

  Podem pensar amb un els tokens com una nova criptomoneda que circula per la xarxa Ethereum, aquesta nova moneda pot tenir camps de dades com quantitat, receptor, creador, nom del token etc. La gràcia és que ens permeten ampliar la funcionalitat de la cadena de blocs mantenint la seguretat que ens proporciona Ethereum, perquè estem utilitzant les verificacions i els nodes què aquesta ens proporciona. Aquests tokens poden seguir un estàndard estipulat i aconseguir propietats encara més interessants, algunes propietats i alguns beneficis de treballar amb tokens:

  - Si són creats mitjançant un estàndard es beneficien de la infraestructura existent d'Ethereum en lloc d'haver de construir una blockchain completament nova per a ells.
  - Per altra banda la creació de nous tokens enforteix l'ecosistema d'Ethereum impulsant la demanda de l'ether, el que fa que tota la xarxa es torni encara més segura.
  - Interoperabilitat. Si tots els tokens creats a la xarxa Ethereum utilitzen el mateix estàndard, aquests tokens seran fàcilment intercanviables i poden treballar fàcilment amb altres aplicacions del mateix ecosistema.

  Anem a esmentar alguns exemples:

  - Si nosaltres tenim una aplicació descentralitzada que utilitza tokens per exemple com moneda d'un joc, podem tenir un sistema on vàries aplicacions utilitzin aquesta moneda, creant així una comunitat més gran per exemple.
  - Un altre exemple pot ser un token que representa un vot com en el nostre cas, la informació del nostre vot, anirà emmagatzemada dins del camp de dades d'una transacció d'Ethereum i el token ens ajudarà a processar les dades de forma més senzilla.

Alguns estàndards de tokens que ha desenvolupat la comunitat d'Ethereum són ERC-20, ERC-223 i ERC-721. Aquests estàndards ja són utilitzats en vàries aplicacions descentralitzades que els usuaris utilitzen com per exemple jocs.

  - En ERC-20 si s'envia un token amb ether a un altre no compatible, la transacció no serà rebutjada perquè el contracte no reconeixerà la transacció entrant. L'ether podria quedar encallat i acabar perdent-se.
  - ERC-223 resol aquest problema rebutjant les transaccions si són incompatibles.
  - Per últim ERC-721 cada token tindrà una característica diferent, seran tokens no fungibles, és a dir no intercanviables entre ells.

Donada aquesta introducció als tokens veiem que tenim àmplies possibilitats i que depenen de les característiques de la votació o del sistema que necessitem ens podem adaptar.

- **Només un vot per persona**

  La primera dificultat que trobem és que a la cadena de blocs no tenim noció d'una persona o d'un individu.

  En una votació tradicional, com hem vist, sabem exactament qui són els nostres diputats o representants. Per a unes eleccions generals, la situació és menys clara i es dedica molt esforç a mantenir les llistes electorals, on votem una representació. A la cadena de blocs veiem que la tasca es torna impossible (molt més complicat de definir.

  Com hem vist en en Ethereum només distingim d'adreces a comptes d'usuaris o de contractes. No és pràctic en una votació demanar una adreça i un vot: Qualsevol atacant podria generar milers d'adreces d'Ethereum i enviar missatges signats per cada un d'ells. Això és indistingible de milers de persones que envien un missatge cadascuna. Aquest tipus d'atac es coneix com a *Sybil*[6]. Per tant, per mitigar els atacs de *Sybil*, hem de tenir una assemblea tancada que administri activament als seus membres,

o un mètode de votació que no compti les adreces. Per a nosaltres aquesta necessitat responia al fet de tenir una entitat autoritat A, la qual ja acceptàvem com a necessària per procedir amb els passos de la votació.

- **Atac amb doble vot**

  En la cadena de blocs ens és impossible votar simultàniament. Que tots els votants votin al mateix temps implicaria que tots els vots siguin part de la mateixa transacció. Això significa que les urnes han d'estar obertes el temps suficient perquè qualsevol pugui votar i en qualsevol ordre.

  Un atac simplificat seria el següent cas:

  1. Alice vota amb el seu token t.
  2. Alice transfereix el seu token t a Bob.
  3. Bob vota amb el token t.

  Les dos comptes han votat amb el mateix token, això és una manipulació del resultat. Per resoldre aquest atac hem d'evitar que Alice pugui enviar el seu token i algunes maneres de fer-ho seria bloquejant el token, permitent votar nombres a un cens i guardar qui ha votat, eliminar el token després de realitzar el vot.

  De nou existeixen moltes possibilitats a l'hora de dissenyar un sistema i les implementacions d'una mateixa alternativa poden ser molt diferents entre ells, per això només comentem la idea que hem de tenir en compte els atacs possibles en el nostre sistema concret.

Capítol 4

Detalls d'implementació

Ena quest capítol comentarem la nostra implementació, tant l'apartat de formalitat criptogràfica com les eines per a programar la solució que es trobaven disponibles en el moment de realitzar aquest treball.

4.1 Disseny

4.1.1 Protocols generals de votació

En la secció de votació en blockchain hem comentat les possibilitats amb que treballem en Ethereum. Ara anem a plantejar la metodologia que hem utilitzat per a decidir quines solucions s'adaptaven més al nostre problema.

Primer hem de tenir en compte els diferents tipus de protocols de votació que tenim disponibles:

1. **Injecció d'estat:** S'injecta l'estat de la votació en el contracte, no necessitem definir tokens. És a dir, en una transacció tx adjuntem les dades de vot en el camp de dades, un cop arriba al contracte aquestes dades es processen dins del contacte.

    - Avantatges: Simplicitat, no necessitem tokens.
    - Inconvenients: Els tokens fungibles poden tenir diferents valors. Un token fungible és un token que no pot ser distingit de cap manera d'un altre token. Un token no fungible significa que té característiques especials i que es poden distingir.

2. **Sense fungibilitat:** Els tokens tenen característiques que els identifiquen, podem per exemple definir tokens que tenen més pes que uns altres. Seria útil en el cas de vots fraccionats com per exemple en una assemblea d'una empresa.

- Avantatges: Simplicitat, n'és un exemple l'estàndard ERC-721.
- Inconvenients: Depenent del tipus de votació no tenen, sentit ja que donarien informació del votant o potser no volem que els vots siguin diferents.

3. **Tokens amb historial i una data de validesa:** Cada token conté informació de la votació i determina quan és vàlid i si ha votat amb anterioritat. És a dir, un token conté una estructura de dades que determina per quines votacions ha passat, això pot ser útil si volem fer un sistema on tinguem múltiples votacions administrades pel mateix contracte.

    - Avantatges: Els tokens tenen un temps de validesa, durant el qual es poden intercanviar perquè són fungibles i podem votar en diferents votacions.
    - Inconvenients: Estructura més complexa de mantenir per al programador, a més complexitat més riscos de cometre erros de seguretat.

4. **Tokens eliminats després d'una votació:** Els tokens no són bloquejats en una votació, sinó que estan pensats per a eliminar-se. Com hem vist, bloquejar els tokens és una manera d'evitar que es manipuli una votació, una altra opció és eliminar aquests tokens.

    - Avantatges: Compatible amb ERC-20 i permet votacions senzilles.
    - Inconvenients: Limita la infraestructura de la votació i l'escalabilitat de l'aplicació.

Per elegir un protocol amb token o injecció d'estat, hem de tenir clares dues situacions en la votació i pensar si volem o no fungibilitat.

- Quan un vot és vàlid?
- Quan podem realitzar l'acció de còmput del resultat?

Si en volem, la següent pregunta és: qualsevol token pot votar o ha de ser un token predeterminat? Les dues situacions que hem de tenir clares són les fases de vot, en la cadena de blocs podem ser conceptes més abstractes que en votacions tradicionals, ja que hem de tenir en compte el comportament d'Ethereum, la seva EVM i la seguretat del sistema.

4.1.2 Selecció d'un protocol pel problema

El protocol que hem definit ens permet aconseguir els objectius que ens hem proposat, els requisits de seguretat i les pautes fonamentals d'un sistema criptogràfic. Primer anem a discutir els protocols de votació que hem vist en l'apartat anterior.

En el nostre cas hem triat un protocol per injecció d'estat, el justifiquem de la següent manera:

- L'objectiu era una votació senzilla, on donades unes propostes els votants elegien entre les possibilitats. Si féssim servir tokens o faríem per tal que les nostres votacions estiguessin controlades per un sol contracte, i de forma similar a aplicacions de sistemes tipus *DAO*, *Decentralized Autonomous-Organization*. L'objectiu seria que amb un sol contracte diverses votacions amb les mateixes característiques poguessin ser autogestionades.
- No requeríem que els nostres vots poguessin ser fraccionals o ponderats.
- No volíem que els vots siguin no fungibles com en un estàndard ERC-721.

Un cop seleccionat aquest primer pas hem d'observar que totes les votacions aniran a un mateix contracte que processarà els vots. Anem pas per pas a avaluar els problemes que es plantegen i utilitzarem la nomenclatura dels objectius:

V votant, A autoritat, v vot, m missatge, D domini.

4.1.3 Primera fase: Confirmació

En la nostra recerca ens hem trobat amb el mateix problema que altres grups de recerca, la figura d'una autoritat A ha sigut impossible d'eliminar del problema. El problema recau en què la nostra idea de votació va en contra del protocol de la cadena de blocs, volem separar un compte de les seves transaccions. Hi ha protocols com Monero[27] que són anònims, utilitzen proves de coneixement zero per tal d'aconseguir això però Ethereum avui en dia no contempla aquesta possibilitat.

Totes les votacions formals en blockchain que s'han realitzat en aquest any, utilitzen aquesta figura central principalment per separar la identitat del votant de la seva compta. Nosaltres a l'igual que altres projectes, reutilitzen aquesta figura per a fer tasques d'administració com finalitzar la fase de confirmació i començar la fase de votació, però això és una conseqüència de què no podem eliminar l'autoritat. Si no tinguéssim autoritat no la voldríem en cap moment, el nostre contracte s'autoregularia i computaria els resultats de forma autònoma.

Alguns projectes actuals han utilitzat mètodes per separar la identitat com signatura d'anell o empremta digital sega, aquestes possibilitats també les hem tingut en compte i això és una bona indicació de que hem estat avançant en el camí correcte.

En la nostra implementació hem utilitzat estats en el contracte per definir quines operacions eren vàlides i quan ho eren, així la infraestructura del contracte està preparada per afegir ampliacions de forma fàcil.

En la fase de confirmació hem optat amb què els votants V puguin registrar-se a la votació i un cop l'autoritat A cregui comenci la següent fase. Durant aquesta fase de confirmació podem realitzar algunes millores al protocol com

per exemple podem acordar mitjançant Deffie-Hellman la clau que utilitzarem per encriptar, es pot bloquejar direccions que intentin atacar la votació o podem confirmar les opcions a votar i que no es puguin modificar en el futur.

4.1.4 Segona fase: Votació

Primer observem que quan votem la informació del vot v ha d'estar situada en una transacció tx per ser avaluada en el contracte. També hem de plantejar-nos que fem amb les dades del votant un cop arriben al contracte.

1. Hem d'encriptar el vot i aquesta encriptació haurà de produir-se fora de la cadena de blocs. En el nostre cas hem escollit ElGamal amb la propietat homomòrfica i ho hem implementat en Java.

2. El contracte pot comprovar que vot forma part del domini i que per tant és vàlid.

3. Quan es realitza la transacció, marquem que un participant ja ha votat i no guardem el seu vot.

4. Un cop tenim un vot vàlid no guardem el vot i l'associem al votant sinó que les dades encriptades les anem sumant i quan tothom ha votat tenim com a resultat la suma de tots els vots. Aquest motiu és el qual ElGamal homomòrfic funciona tan bé en el nostre cas, ja que d'una manera fàcil afegim una petita capa de protecció al sistema i l'encriptació i desencriptació és ràpida.

4.1.5 Tercera fase: Recompte

Quan l'autoritat A finalitza la fase de votació, pot agafar el resultat encriptat i realitzar el còmput final fora de la blockchain. El resultat com hem dit és la suma dels vots, anem a explicar com funciona el còmput.

Imaginem que les possibilitats per a votar són, si, no o deixar vot en blanc. També limitem el nombre de votants a 100 per exemple. El nostre objectiu és donar un valor numèric a cada vot de tal forma que després ho puguem factoritzar. La manera seria la següent:

- Si: 101, nombre de votants + 1.
- No: 1, si tothom vota No el resultat seria 100.
- Blanc: 0

Si tothom vota No el resultat seria 100, si tothom vota Si el resultat seria 10100. Donat un resultat qualsevol només hem de trobar els factors per saber el nombre de si i el nombre de no. Aquesta és una manera enginyosa de modelitzar

els missatges i facilitar els còmputs.

Un cop l'autoritat sap el resultat la pot publicar en la cadena de blocs a la resta de participants i de nou un sistema de validacions podria verificar la seva correctesa de manera que es protegeixin els vots privats de cada participant i només és sàpigues que el resultat final és correcte.

Com observem és una tasca realment complicada treballar a contra natura del protocol i avui en dia no hi ha una solució òptima. Rellevant la seguretat que volem a proves de correctesa o de coneixement nul estem incrementant la dificultat de la implementació.

4.2 Aplicació

La implementació es pot separar en tres mòduls, els quals hem intentat escollir els que millor s'adaptin entre ells. Per a que siguin prou flexibles per ser expandits i també que la interacció tingui poques dependències per tal de poder reutilitzar algun d'aquests mòduls en el futur.

Els tres mòduls que comentem són els que s'encarreguen de:

- Client web: Hem implementat una petita aplicació amb Angular[22] de votació a forma de prototip, no és interactiu però és una bona visualització de les possibilitats que oferim.
- Ethereum: Utilitzant Truffle[23], simulem la cadena de blocs d'Ethereum i despleguem els nostres contractes.
- Criptografia: Aquest mòdul s'encarrega de realitzar l'encriptació, desencriptació i còmput del resultat de la votació. No detallarem res en la memòria ja que és la implementació dels algoritmes que hem vist al capitul preliminar.

En aquesta secció comentarem les implementacions que hem realitzat per tal de fer el prototip, aquesta solució està lluny de ser la desitjada i no es comporta com hem explicat en el disseny del sistema. La dificultat recau en què treballar amb nombres tan elevats de bits en la cadena de blocs ara com ara no hi ha cap llibreria que ho faciliti i implementar-ho requeriria més temps.

Per tant, per tal de poder implementar les nostres idees principals i comprovar el comportament en la blockchain, no hem implementat les validacions en cap contracte sinó que el mòdul d'Ethereum només simula les fases de votació i les seqüències que s'haurien de seguir però no valida cap resultat. Aquestes validacions i proves de coneixements ja són utilitzades i conegudes en el món de la votació electrònica. Per tant podem a priori assegurar la formalitat del protocol tot i que en una implementació real pot ser que hi hagi petits matisos.

4.2.1 Metamask

MetaMask[25] és la forma més fàcil d'interactuar amb aplicacions descentralitzades en un navegador. És una extensió per Chrome o Firefox que es connecta a una xarxa Ethereum sense executar un node complet. Pot connectar-se a la xarxa principal d'Ethereum, a qualsevol de les xarxes de prova (Ropsten, Kovan i Rinkeby)[28], o a una cadena de blocs local com la creada per Ganache[24] o Truffle[23].

En el nostre cas, atès que hem implementat un client web que s'executa en navegadors ha sigut una decisió fàcil utilitzar Metamask, també ens permet de forma fàcil ajustar els costos de gas de la transacció.

A continuació mostrem dues imatges de com s'utilitza:

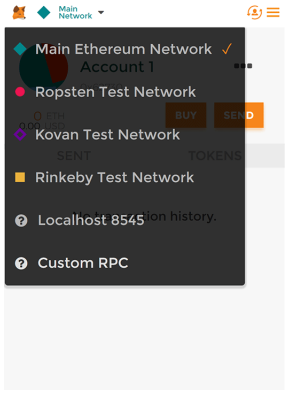


Figura 4.1: Ens permet interactuar amb diferents cadenes de blocs.

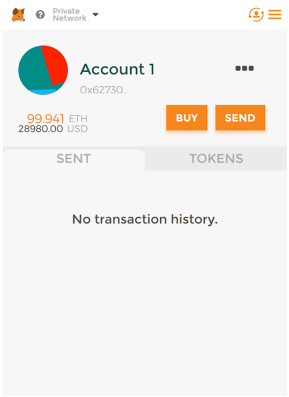


Figura 4.2: Ens permet generar contes i realitzar transaccions.

4.2.2 Truffle

Per tal desenvolupar hem utilitzat Truffle [23], un entorn que agilitza els tests i ens proporciona una cadena de blocs local utilitzant la màquina virtual Ethereum (EVM), algunes de les característiques que disposa:

- Compilació, enllaç, desplegament i gestió binària de contractes intel·ligents incorporats.
- Proves dels contractes automatitzades per a un desenvolupament ràpid.
- Implementació i migracions extensibles i programables.
- Gestió de xarxes per al seu desplegament.
- Gestió de paquets amb EthPM i NPM, utilitzant l'estàndard ERC190.

L'estructura de fitxers que segueix és la següent i tot seguit anem a comentar quina utilitat tenen els arxius que es generen:

- contracts/: Directori pels contractes Solidity.
- migrations/: Directori pels arxius de desplegament.
- test/: Directori pels tests, en el nostre cas els hem realitzat per comprovar el correcte desenvolupament de les fases durant la votació.
- truffle.js: Un arxiu de configuració de l'entorn.

El desenvolupament en Truffle disposa de vàries fases, primer hem d'escriure els contractes en la carpeta *contracts/*. Al compilar a part de verificar la correctesa dels contractes segons la versió de Solidity en què treballem, Truffle genera artefactes per a cada un d'aquests contractes per a facilitar el desplegament. Anem a observar el nostre arxiu principal de migració per a entendre el que estem fent sobre:

```
var ECCMath = artifacts.require("./ECCMath.sol");
var Owned = artifacts.require("./Owned.sol");
var Voting = artifacts.require("./Voting.sol");
var Secp256k1 = artifacts.require("./Secp256k1.sol");
var Set = artifacts.require("./Set.sol");

module.exports = function(deployer) {

  deployer.deploy(ECCMath);
  deployer.link(ECCMath, Secp256k1);

  deployer.deploy(Secp256k1);
  deployer.link(Secp256k1, Voting);
```

```
  deployer.deploy(Owned);
  deployer.link(Owned, Voting);

  deployer.deploy(Set);
  deployer.link(Set, Voting);

  deployer.deploy(Voting);

};
```

Sobre aquest exemple veiem que disposem d'uns quants contractes en els quals per exemple tenim llibreries matemàtiques per la part criptogràfica com *ECCMath.sol* i altres utilitats. El contracte principal és *Voting.sol* i el nostre objectiu és el de desplegar-ho correctament sobre la cadena de blocs. Bàsicament utilitzem dos mètodes:

- *deployer.link*(*contract*1, *contract*2);
- *deployer.deploy*(*contract*2)

Amb el mètode *link* resolem les dependències d'importació que hi ha en els contractes, primer despleguem un contracte com per exemple *Owned.sol* després relacionem aquest contracte amb *Voting.sol* i això fem per totes les dependències necessàries, per últim despleguem el contracte *Voting.sol*.

Un cop hem migrat podem realitzar el desplegament sobre la nostra cadena privada de blocs la qual se'ns inicialitza automàticament amb 10 comptes d'Ethereum com podem veure en aquesta sortida de consola.

```
Truffle Develop started at http://localhost:9545/

Accounts:
(0) 0x627306090abab3a6e1400e9345bc60c78a8bef57
(1) 0xf17f52151ebef6c7334fad080c5704d77216b732
(2) 0xc5fdf4076b8f3a5357c5e395ab970b5b54098fef
(3) 0x821aea9a577a9b44299b9c15c88cf3087f3b5544
(4) 0x0d1d4e623d10f9fba5db95830f7d3839406c6af2
(5) 0x2932b7a2355d6fecc4b5c0b6bd44cc31df247a2e
(6) 0x2191ef87e392377ec08e7c08eb105ef5448eced5
(7) 0x0f4f2ac550a1b4e2280d04c21cea7ebd822934b5
(8) 0x6330a553fc93768f612722bb8c2ec78ac90b3bbc
(9) 0x5aeda56215b167893e80b4fe645ba6d5bab767de

Private Keys:
(0) c87509a1c067bbde78beb793e6fa76530b6382a4c0241e5e4a9
(1) ae6ae8e5ccbfb04590405997ee2d52d2b330726137b875053c3
(2) 0dbbe8e4ae425a6d2687f1a7e3ba17bc98c673636790f1b8ad9
```

```
(3) c88b703fb08cbea894b6aeff5a544fb92e78a18e19814cd85da
(4) 388c684f0ba1ef5017716adb5d21a053ea8e90277d086833751
(5) 659cbb0e2411a44db63778987b1e22153c086a95eb6b18bdf89
(6) 82d052c865f5763aad42add438569276c00d3d88a2d062d36b2
(7) aa3680d5d48a8283413f7a108367c7299ca73f553735860a87d
(8) 0f62d96d6675f32685bbdb8ac13cda7c23436f63efbb9d0770e
(9) 8d5366123cb560bb606379f90a0bfd4769eecc0557f1b362dc3
```

El nostre contracte *Voting.sol* modelitza la votació amb diferents estats, els quals per a cada funció que es pot cridar comprovem que estiguem en un estat vàlid. Aquest pot ser el detall més interessant de la implementació i val la pena comentar-ho. Per escriure aquest codi hem seguit les indicacions d'Ethereum per a escriure codi segur [10] i evitar atacs típics [11]. Tenim un modificador inState el qual podem afegir als mètodes del contacte, en aquest exemple necessitem estar en l'estat de *SIGNUP* i si no ens trobem en aquell estat és la EVM qui rebutja el codi de forma segura.

```
  modifier inState(State expected) {
    require(state == expected);
    _;
}

  function setStateToVote() inState(State.SIGNUP)
  public onlyOwner {
    require(numberOfParticipants == numberOfRegistered);
    state = State.VOTE;
}
```

Una de les recomanacions a l'escriure codi en Solidity és que aquest sigui el més senzill possible i utilitzar al màxim les capacitats del llenguatge d'interactuar amb la EVM. Podríem haver evitat aquest problema utilitzant condicionals, però no seria la màquina virtual qui arregles el problema donant lloc a possibles llocs d'atac i errors de programació.

El codi final és compacte i senzill, tan d'interpretar com d'escriure, seguint les indicacions per evitar atacs no només fem codi segur sinó més usable.

4.2.3 Client web

Sobre el client no comentarem gaire, només és per donar una idea de les possibilitats, en el nostre exemple vam agafar per exemple una aplicació similar a *change.org* on la gent voti per temes medi-ambientals, salut, economia... En la figura 4.3 podem veure un exemple de dashboard.

La idea és que qualsevol persona pugui crear una votació i la podem representar mitjançant un token en la blockchain, un camp d'aquest token pot ser la categoria per exemple, el nostre codi pot ser senzill i deixem que sigui el client

que interpreti que significa cada camp. En la figura 4.4 podem veure un exemple de formulari de votació, on a part de consultar l'estat de la cadena de blocs en temps real 4.5, podem veure tota la informació pública.

Aquest client era una fase de treball original on contemplàvem dedicar-nos més a fer una aplicació descentralitzada. Igualment ens ha servit per interactuar amb la cadena de blocs i poder veure de primera mà com adaptar aquest tipus de sistema i com ho podríem utilitzar en algunes arquitectures.

Avui en dia les aplicacions que utilitzen cadena de blocs només la utilitzen per tasques molt concretes i formen part d'un sistema amb servidors tradicionals, microserveis, bases de dades, etc. Nosaltres estem d'acord en el fet que la cadena de blocs és una eina complementaria i que ha de ser utilitzada quan de veritat aporti una solució i sigui necessari.

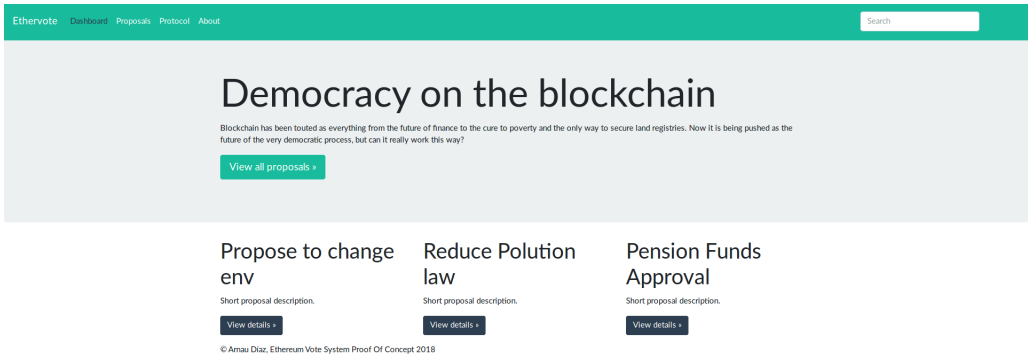


Figura 4.3: Dashboard del client web.

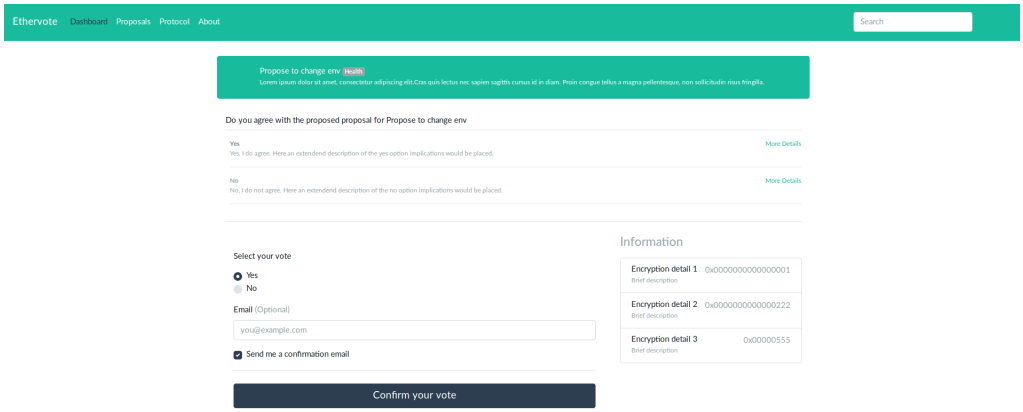


Figura 4.4: Exemple de formulari de votació.

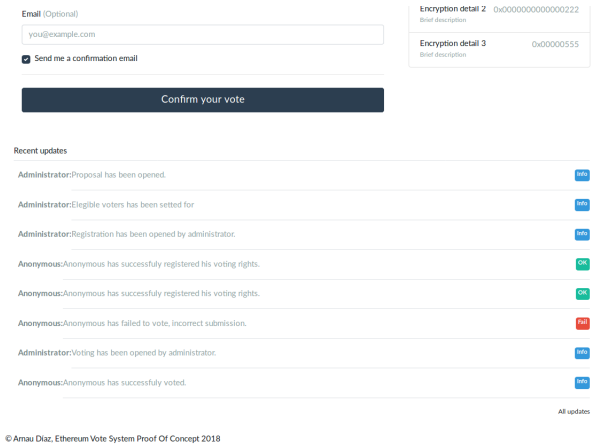


Figura 4.5: Exemple dades en temps real.

Capítol 5

Conclusions

Dividirem les conclusions en tres seccions, per un costat parlarem dels objectius assolits trobats en l'àmbit de disseny d'un sistema de votació, comentarem la implementació que l'acompanya i els problemes que ens hem trobat i per últim comentarem les línies de futur treball.

5.1 Disseny

En la part criptogràfica teòrica del treball, hem comentat que hi ha una gran varietat d'algorismes i possibles implementacions de protocol que durant els últims anys s'han anat implementant i el dia d'avui encara és un camp en què es fa recerca. Pel que fa al desenvolupament d'un treball de fi de grau podem assegurar que hem realitzat una bona metodologia de treball a causa del fet que ens hem anat adaptant a les limitacions que ens hem anat trobat.

Aquestes limitacions són les normals quan es treballa amb tecnologia amb tan poques eines disponibles i s'han fet pocs treballs o estudis reals. Només s'ha de veure la quantitat d'avanços que hi ha en 1 any on realitzar una tasca originalment podia ser un treball de setmanes que es redueix a hores quan hi ha una infraestructura més madura.

El nostre sistema de votació s'ha anat construint en petites passes de tres fases:

1. Traslladar el concepte de votació a blockchain.

2. Analitzar els problemes de seguretat que suposaria tant a possibles atacs.

3. Buscar solucions que poguessin ser implementades o ens proporcionessin el nombre més gran d'avantatges, en el nostre cas validacions, proves de coneixement, etc.

Aquestes iteracions les hem anat desenvolupant estudiant els primers exemples de votació en blockchain que s'havien fet, que en el moment no eren gaires, per totes les passes d'una votació tradicional, la fase preliminar, la votació i el recompte.

Com hem anomenat en els preliminars un sistema ha de poder ser formalitzat sense tenir en compte la seva implementació. Els mètodes d'encriptació que hem acabat utilitzant sabem que són mètodes segurs i era la nostra tasca no comprometre la seguretat un cop aquesta votació es dugués a terme en la cadena de blocs. Com que en blockchain els contractes són públics, els participants saben exactament quins seran els passos i com s'efectuaran, si hi haurà limitacions de temps, les opcions a escollir i tenen l'opció de no participar en una votació que no creguin correcta.

La conclusió més important que podem treure del disseny ha sigut la mateixa a què han arribat altres grups de recerca. No hem sigut capaços de desfer-nos de la figura d'una autoritat en la votació. Aquesta autoritat la podríem intentar minimitzar al màxim i això afectaria el protocol de votació i com s'interactuaria amb els contractes, però avui en dia no som capaços d'eliminar-la completament.

Els nostres sistemes de votació tradicionals són anònims en el sentit que no sabem qui ha votat què, això va en contra del protocol d'Ethereum. Si no hi ha cap canvi en el protocol que permeti per exemple signatures d'anell o sigui una cadena de blocs anònima, encara no tenim manera de simular-ho sense tenir un punt de centralització que comprometi d'alguna manera la seguretat dels participants.

Una possible alternativa és invertir els esforços a crear una blockchain que compleixi les característiques que necessitem, però si només s'utilitza per a una aplicació tan concreta com és la votació potser no tenim un grau de seguretat suficientment gran com per garantir l'estabilitat.

5.2 Implementació

En la implementació no hem arribat a assolir completament els nostres objectius pel fet que no hem sigut capaços en el temps que disposàvem de programar una solució exacta al nostre protocol dissenyat. Hem assolit objectius parcials com per exemple:

- Estudi i treball en blockchains locals.
- Estudi i treball amb aplicacions descentralitzades.
- Interactuar amb la cadena de blocs des de l'aplicació.
- Estudi d'atacs bàsics en Ethereum.

- Exploració de la cadena de blocs i de les traces que hes deixen.
- Implementació d'una api d'encriptació amb ElGamal.
- Desenvolupament d'un prototip de client web.

Algunes conclusions que podem treure són que el desplegament i automatització de contractes avui en dia està ben controlat, empreses com IBM ho permeten en els seus serveis i la *linuxfundation* també dona suport a projectes com Hyperledger o Infura.

És fàcil interactuar amb les cadenes de blocs mitjançant peticions *RPC*, *Remote Procedure Call* les quals poden ser asíncrones, plugins com Metamask faciliten la interacció amb els navegadors tradicionals però tenen el gran problema que són punts d'atac. Metamask de fet ha sigut atacat en el passat per grups organitzats comprometen la seguretat de moltes carteres d'Ethereum. Llavors si evitem el servei de tercers i els implementem nosaltres, podem fer-nos responsables de la nostra seguretat.

Ens hem trobat amb un problema que originalment no consideràvem que fos tant complex. Aquest és programar en Solidity una llibreria matemàtica que ens permeti realitzar les validacions criptogràfiques del nostre sistema. El problema recau en què estem treballant amb massa bits i requeriríem programar els sistemes per a la nostra solució concreta en llenguatge de baix nivell assemblador.

En la nostra fase de recerca hem trobat algun article que ha treballat en aquestes línies d'assolir el sistema de manera funcional[29], però són implementacions concretes per a cada mètode d'encriptació i no és possible reutilitzar el codi.

Per això el nostre codi en Solidity només ha servit per dissenyar les fases de votació de forma segura a atacs i poder tenir una visió real de l'estat de l'art actual. Hem escrit tests unitaris per comprovar la validesa del que programàvem i podem concloure també que l'estat actual dels entorns de desenvolupament és vàlid per programar i tenim suport per a realitzar proves de forma potent, el problema que ens podem trobar si treballéssim en un altre projecte és infravalorar el pes de la computació, ja que l'estat de llibreries està encara en una fase molt poc madura.

Per últim podem concloure que l'objectiu d'aquest treball, la criptografia aplicada, és probablement un dels temes més complexos en Ethereum, tant pel disseny com per la implementació. Sent també la part d'implementació de llibreries computacionals costoses un factor complex en qualsevol tipus de cadena de blocs amb les limitacions que aquestes tenen actualment.

5.3 Treball futur

Després de la realització d'aquest treball obrim varies línies de treball futur, per un costat tenim ampliacions d'aquests, per un altre propostes de recerca o TFG alternatius de temàtica similar al que hem treballat. Anomenem possibles ampliacions:

- Substituir les eines de tercers per eines pròpies.
- Acabar la interacció amb la cadena de blocs del client, podem integrar-ho en un servidor.
- Treballar en la llibreria matemàtica de baix nivell escrita en Solidity.
- Fer un estudi més detallat dels possibles atacs en el nostre sistema de votació.

Altres treballs interessants amb poca documentació:

- Programar el nostre protocol de votació en un llenguatge d'alt nivell com Java i implementar totes les capacitats matemàtiques que hem proposat, proves de correctesa, proves de coneixement zero, etc.
- Desenvolupar una aplicació descentralitzada amb la cadena de blocs d'Ethereum.
- Desenvolupar aplicacions descentralitzades que requereixin d'estructures de tokens més complexes.
- Estudiar en profunditat els atacs en la cadena de blocs.
- Aplicar la tecnologia blockchain a l'IOT, $Internet\ Of\ Things$, el qual té un gran futur en la industria.
- Realitzar modificacions al protocol d'Ethereum.
- Treballar en un projecte encarat als nodes de la xarxa i la minera dels blocs.
- Estudiar les noves criptomonedes com $IOTA$ que utilitzen estructures que no són cadenes de blocs.

Capítol 6

Bibliografia

[1] A. M. Antonopoulos *Mastering Bitcoin*. O'Reilly, 1993.

[2] W. Dai *b-money*. http://www.weidai.com/bmoney.txt 1998.

[3] W. Diffie, M. Hellman *New Directions in Cryptography* . 1976.

[4] T. ElGamal. *A public key cryptosystem and a signature scheme based on discrete logarithms.*. IEEE trans. Inform. Theory, 1985.

[5] S. Nakamoto *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2009.

[6] J.M Seigneur, Gray, Alan, Jensen, C. Damsgaard *Trust Transfer: Encouraging Self-recommendations Without Sybil Attack*

[7] R.L. Rivest, A. Shamir, L. Adleman *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*

[8] M. Wenbo *Modern Cryptography: Theory and Practice*. Prentice Hall PTR, 2003.

[9] Namecoin *https://namecoin.org/*

[10] Ethereum Smart Contract Security Best Practices https://consensys.github.io/smart-contract-best-practices/

[11] Known attacks https://consensys.github.io/smart-contract-best-practices/known-attacks/

[12] Computational complexity https://en.wikipedia.org/wiki/Computational-complexity-theory

[13] Discrete logarithm https://en.wikipedia.org/wiki/Discrete_logarithm

[14] Hash Function *https://en.wikipedia.org/wiki/Hash_function*

[15] One way function *https://en.wikipedia.org/wiki/One-way-function*

[16] Blockchain 3.0: A Natural Evolution?
https://medium.com/datadriveninvestor/blockchain-3-0-a-natural-evolution-cfa5487254c6

[17] Masked Ballot Voting for Receipt-Free Online Elections
https://link.springer.com/chapter/10.1007/978-3-642-04135-8_2

[18] MVP: An Efficient Anonymous E-Voting Protocol
https://ieeexplore.ieee.org/document/7842019/

[19] A Secure and Optimally Efficient Multi-Authority Election Scheme *http://www.win.tue.nl/ berry/papers/euro97.pdf*

[20] Can Elgamal be made additively homomorphic and how could it be used for E-voting?
https://crypto.stackexchange.com/questions/3626/can-elgamal-be-made-additively-homomorphic-and-how-could-it-be-used-for-e-voting/3630

[21] Multiplicative vs Additive Homomorphic ElGamal
https://nvotes.com/multiplicative-vs-additive-homomorphic-elgamal/

[22] Angular *https://angular.io/*

[23] Truffle suite *https://trufflframework.com/*

[24] Ganache: Personal blockchain for Ethereum development
https://github.com/trufflesuite/ganache

[25] Metamask *https://metamask.io/*

[26] Ethereum *https://github.com/ethereum/wiki/wiki/White-Paper*

[27] Monero *https://whitepaperdatabase.com/monero-xmr-whitepaper/*

[28] Ethereum network *http://ethdocs.org/en/latest/network/index.html*

[29] A Smart Contract for Boardroom Voting with Maximum Voter Privacy *http://fc17.ifca.ai/preproceedings/paper_80.pdf*