

Sistema de votación electrónica basado en blockchain

Pedro García Cereijo

Tutores: Paula Fraga Lamas y Tiago Manuel Fernández Caramés

Curso 2023/2024

Pedro García Cereijo

Sistema de votación electrónica basado en blockchain

Trabajo de Fin de Máster. Curso 2023/2024

Tutores: Paula Fraga Lamas y Tiago Manuel Fernández Caramés

Máster Inter-Universitario en Ciberseguridad

Universidade da Coruña

Facultade de Informática

Camiño do Lagar de Castro, 6

15008, A Coruña

Abstract

This Master's Thesis presents a robust and secure voting system based on the Ethereum blockchain, integrated with oracles, IPFS (InterPlanetary File System) and OrbitDB. The proposed system leverages the decentralized nature of blockchain technology to ensure transparency and immutability in the voting process. Oracles are used to connect the blockchain with the outside world, allowing for the integration of external data and real-time information verification. IPFS is employed for decentralized storage, ensuring that all voting data are stored securely and efficiently, maintaining accessibility and resilience against censorship or data loss. OrbitDB, a decentralized database built on IPFS, enhances the system by providing scalable and distributed data storage, allowing for the efficient handling of large volumes of voting data and improving the overall performance and reliability of the system. This approach addresses the main challenges of security, transparency and scalability in electronic voting systems, aiming to create more reliable and resilient electoral processes.

Keywords — Blockchain, Ethereum, IPFS, OrbitDB, Voting

Resumen

Este Trabajo de Fin de Máster presenta un sistema de votación robusto y seguro basado en la blockchain de Ethereum, integrado con oráculos, IPFS (InterPlanetary File System) y OrbitDB. El sistema propuesto aprovecha la naturaleza descentralizada de la tecnología blockchain para garantizar transparencia e inmutabilidad en el proceso de votación. Se utilizan oráculos para conectar la blockchain con el mundo exterior, permitiendo la integración de datos externos y la verificación en tiempo real de la información. IPFS se emplea para el almacenamiento descentralizado, asegurando que todos los datos de votación se almacenen de manera segura y eficiente, manteniendo la accesibilidad y la resistencia contra la censura o la pérdida de datos. OrbitDB, una base de datos descentralizada construida sobre IPFS, mejora el sistema al proporcionar almacenamiento de datos escalable y distribuido, permitiendo el manejo eficiente de grandes volúmenes de datos de votación y mejorando el rendimiento y la fiabilidad general del sistema. Este enfoque aborda los principales desafíos de seguridad, transparencia y escalabilidad en los sistemas de votación electrónica, buscando crear procesos electorales más confiables y resistentes.

Palabras clave — Blockchain, Ethereum, IPFS, OrbitDB, Votación

Agradecimientos

A mis padres, por su inquebrantable apoyo en cada paso del camino durante estos últimos seis años. Gracias por escucharme hablar durante horas sobre ciberseguridad y blockchain, a pesar de no entender siempre lo que decía.

A Tiago y Paula, gracias por darme la oportunidad de formar parte de este increíble equipo, por guiarme y por ayudarme a crecer profesionalmente. Su apoyo y orientación han sido cruciales en este proyecto.

A Ana, gracias por acompañarme en todo momento, por mostrar un interés genuino en mi trabajo y por ser un apoyo constante en cada etapa de este proyecto.

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.2.1. Metodología	3
1.3. Estructura de la memoria	4
2. Análisis de la situación actual	7
2.1. Trabajo relacionado	7
2.1.1. Sistemas de votación actuales	7
2.1.2. Sistemas de votación basados en blockchain	8
2.1.3. Análisis de la tecnología blockchain	10
2.1.4. Análisis sobre las mejoras de seguridad que se pueden imple- mentar en el sistema	12
2.2. Tecnologías base	12
2.2.1. Ethereum	12
2.2.2. Node.js	13
2.2.3. Solidity	13
2.2.4. MetaMask	14
2.2.5. Hardhat	14
2.2.6. IPFS	15
2.2.7. OrbitDB	15
3. Diseño del sistema	17
3.1. Definición de los requisitos	17
3.1.1. Requisitos no funcionales	17
3.1.2. Requisitos funcionales	18
Votantes:	18
Administradores:	19
3.1.3. Casos de uso	20
Registro	20
Autenticación	21
Seleccionar votación	21
Votar	23

Comprobar los datos de sus votos	23
Crear votación	24
Cerrar votación	24
Obtener resultados	25
3.2. Arquitectura del sistema	26
4. Implementación del sistema	29
4.1. Smart Contract	29
4.2. Oráculo	33
4.3. OrbitDB	43
4.4. Front-End	46
Política de contraseñas	51
4.4.1. Comunicación con el oráculo	53
5. Pruebas	55
5.1. Smart contract	55
5.2. Oráculo	56
5.3. Nodos de OrbitDB	57
5.4. Front-End	59
6. Resultados	61
6.1. Evaluación de la seguridad	61
6.1.1. Aspectos a tener en cuenta	61
6.1.2. OrbitDB	62
6.1.3. Oráculo	63
Descentralización del oráculo	64
6.2. Posibles ataques en redes blockchain	66
6.2.1. Ataque del 51%	66
6.2.2. Ataques MitM	67
6.2.3. Ataques cuánticos	69
7. Conclusiones	71
7.1. Limitaciones y trabajo futuro	72
7.1.1. Smart Contract	72
7.1.2. Oráculo	72
7.1.3. OrbitDB	73
7.1.4. Front-End	74
7.1.5. Blockchain	75
7.1.6. Securitización post-quántica	76

A. Anexo - Planificación	77
A.1. Planificación temporal	77
A.1.1. Ciclo 1 - Planificación inicial	77
A.1.2. Ciclo 2 - Diseño y desarrollo de los componentes independientes	77
A.1.3. Ciclo 3 - Integración y pruebas de integración	78
A.1.4. Ciclo 4 - Análisis del sistema	78
A.1.5. Ciclo 5 - Despliegue	79
A.1.6. Ciclo 6 - Realización de la memoria	79
A.2. Costes estimados	80
B. Anexo - Instalación y walkthrough	81
B.1. Instalación	81
B.1.1. Hardhat	81
B.1.2. OrbitDB	84
B.1.3. Oráculo	85
B.1.4. Front-End	86
B.2. Walkthrough	86
Bibliografía	93

Introducción

En este capítulo se presentará una visión general del proyecto, su motivación original y los objetivos que se pretenden alcanzar con este trabajo.

El objetivo de este proyecto es diseñar e implementar un sistema de votación electrónica basado en blockchain que permita votar de forma segura y anónima, y que sea resistente a diversos tipos de ataques que pueda sufrir la red.

1.1 Motivación

Los sistemas de votación juegan un papel fundamental en la sociedad moderna; sin embargo, muchos de ellos no han cambiado desde hace décadas. Por ello, la implementación de tecnologías avanzadas como blockchain en los sistemas de votación electrónica representan una oportunidad única para mejorar la seguridad, transparencia y confianza en los procesos electorales. Este TFM (Trabajo de Fin de Máster) del Máster Inter-Universitario en Ciberseguridad se centra en la exploración y desarrollo de un sistema de votación electrónica basado en blockchain, motivado por varios factores detallados a continuación.

En primer lugar, la tecnología blockchain ofrece un nivel de seguridad sin precedentes al proporcionar un registro inmutable y a prueba de manipulaciones de cada voto. Al almacenar cada voto de forma segura en la blockchain y permitir su verificación por cualquier persona, se elimina la posibilidad de fraudes o manipulaciones en el proceso electoral, garantizando así la integridad y la transparencia de las elecciones.

Además, los sistemas de votación basados en blockchain son altamente eficientes en comparación con los métodos tradicionales. La velocidad con la que los votos pueden ser registrados y almacenados en la blockchain es significativamente mayor que los sistemas de votación convencionales, que pueden tomar días o incluso semanas para ser contabilizados. Esta rapidez no solo facilita el proceso de votación, sino que también permite una gestión más ágil y una supervisión en tiempo real por parte de las autoridades electorales. La rapidez en la contabilización de los

votos también reduce el período de incertidumbre post-electoral, contribuyendo a una transición más fluida y menos conflictiva entre los resultados preliminares y los finales.

Otro punto a favor es la capacidad de los sistemas de votación basados en blockchain para proporcionar autenticación de votantes de manera segura. Al utilizar esta tecnología, se garantiza que solo los votantes autorizados puedan emitir su voto, evitando así el riesgo de fraudes electorales o votos duplicados. Esto se logra mediante la verificación y validación segura de la identidad de los votantes, lo que refuerza la confiabilidad del proceso electoral en su totalidad.

Finalmente, los sistemas de votación basados en blockchain proporcionan una mayor transparencia que los sistemas de votación tradicionales debido a la naturaleza distribuida de la blockchain. Todos los votos se almacenan de forma segura en la blockchain, lo que permite a cualquiera verlos y verificarlos. Esto elimina la posibilidad de elecciones amañadas o manipuladas, ya que cualquiera puede verlas y verificar que sean precisas y auténticas. La transparencia adicional no solo aumenta la confianza del público en el sistema electoral, sino que también puede fomentar una mayor participación ciudadana, ya que los votantes pueden estar seguros de que su voto será contado de manera justa y precisa.

1.2 Objetivos

El objetivo de este trabajo es diseñar un sistema de votación electrónica basado en blockchain utilizando tecnologías actuales que permita a los usuarios votar y participar en elecciones respetando en todo momento la seguridad, privacidad y anonimato de los usuarios. Este sistema deberá garantizar que cada voto sea registrado de manera inmutable y verificable, eliminando así cualquier posibilidad de manipulación o fraude electoral. Además, nos centraremos en la seguridad del sistema global, buscando que sea resistente a posibles ataques.

Los objetivos a cumplir son los siguientes:

- Respetar la privacidad de cada votante, garantizando que su identidad sea anónima y protegiendo su derecho al voto secreto.
- Permitir que cada voto sea verificable tanto por la administración como por cualquier persona que desee realizar un recuento por su cuenta.

- Asegurar la seguridad del sistema para que sea invulnerable a modificaciones por parte de atacantes, evitando la adición de votos no legítimos.
- Asegurar la modularidad del sistema, promoviendo siempre la descentralización de los componentes que lo conforman. Esto facilitará la escalabilidad y la adaptabilidad del sistema, permitiendo su implementación en diversos contextos electorales y su evolución a medida que surjan nuevas tecnologías y necesidades.

Asimismo, la idea de este trabajo es que el código pueda ajustarse en función de los escenarios en los que se despliegue el sistema. De esta forma, este sistema podrá ser utilizado para diferentes tipos de votaciones y adaptado según el número de nodos votantes y la red en la que se implemente.

1.2.1 Metodología

El desarrollo de este trabajo estuvo dividido en diferentes fases, cada una con un enfoque específico y esencial para la consecución del objetivo final.

La primera fase del trabajo se centró en una exhaustiva investigación del estado del arte y un análisis de las tecnologías actuales que podrían implementarse en el sistema. Se estudiaron diversas soluciones de votación electrónica existentes y se evaluaron sus ventajas y desventajas, con un énfasis particular en aspectos de descentralización y seguridad.

La segunda fase del proyecto consistió en el diseño e implementación del sistema de votación electrónica basado en blockchain. En esta etapa, se definieron los requisitos técnicos y funcionales del sistema, y se seleccionaron las herramientas y plataformas más adecuadas para su desarrollo. El diseño se enfocó en asegurar la integridad y confidencialidad del voto, así como en garantizar la accesibilidad y facilidad de uso para los votantes. Posteriormente, se llevó a cabo la implementación del sistema, donde se construyeron los componentes necesarios.

La fase final del trabajo se dedicó a analizar la seguridad del sistema implementado. Se realizaron pruebas exhaustivas para identificar posibles vulnerabilidades y evaluar la robustez del sistema frente a diferentes tipos de ataques. Como parte de esta fase, también se diseñaron futuras mejoras que podrían implementarse en el sistema, basadas en los resultados obtenidos durante la evaluación y en las tendencias emergentes en la tecnología blockchain y la ciberseguridad.

1.3 Estructura de la memoria

Los siguientes capítulos de esta memoria se estructuran como sigue:

- **Capítulo 2. Análisis de la situación actual:** En este capítulo se expone la situación actual del ámbito de votación utilizando blockchain. Se lleva a cabo un análisis detallado de los sistemas de votación en blockchain que han sido usados hasta la actualidad, identificando sus características, fortalezas y debilidades. Además, se exploran las posibles mejoras de seguridad que se pueden implementar para optimizar estos sistemas, y se realiza un análisis exhaustivo de las tecnologías básicas que se utilizan, proporcionando una comprensión profunda de los componentes fundamentales del sistema.
- **Capítulo 3. Diseño del sistema:** Este capítulo detalla el trabajo realizado en el desarrollo del sistema. Se comienza con el diseño del sistema a nivel funcional, describiendo cómo se estructuran y organizan los diferentes componentes y módulos para cumplir con los requisitos establecidos. A continuación, se aborda la arquitectura del sistema, proporcionando una visión clara de su estructura y de cómo interactúan los distintos elementos.
- **Capítulo 4. Implementación del sistema:** En este capítulo se explica el proceso de implementación, detallando las herramientas y metodologías utilizadas para la implementación de cada uno de los componentes que conforman el sistema final.
- **Capítulo 5. Pruebas:** En este capítulo se presentan las pruebas realizadas para verificar el correcto funcionamiento de cada uno de los componentes del sistema final, asegurando que cumplen con los objetivos y requisitos definidos.
- **Capítulo 6. Resultados:** En este capítulo se procede a evaluar la seguridad del sistema implementado. Se realiza un análisis detallado de la seguridad a nivel de cada componente del sistema, identificando posibles vulnerabilidades y medidas de mitigación. Además, se lleva a cabo un análisis sobre los posibles ataques que podría sufrir el sistema, evaluando cómo el sistema desarrollado se posiciona en términos de seguridad.
- **Capítulo 5. Conclusiones:** El capítulo final presenta las conclusiones del trabajo realizado. Se ofrece un resumen de todo el trabajo, destacando los logros

y los principales resultados obtenidos. Además, se realiza un estudio del impacto de estos resultados, evaluando su relevancia y aplicabilidad en el contexto de votación utilizando blockchain. Finalmente, se incluye una sección dedicada al trabajo futuro, donde se identifican áreas de mejora y posibles líneas de investigación que podrían explorarse para continuar avanzando en este campo.

- **Anexo A. Planificación:** En este anexo se detalla la planificación realizada para el desarrollo del trabajo, así como un estudio aproximado de sus costes relacionados.
- **Anexo A. Instalación y despliegue:** En este anexo se detallan los pasos a seguir para instalar y desplegar el sistema de votación implementado, así como un walkthrough del funcionamiento normal del sistema.

Análisis de la situación actual

2.1 Trabajo relacionado

2.1.1 Sistemas de votación actuales

Antes de hablar de sistemas de votación, es necesario definir qué es un voto. Un voto es la forma en la que un individuo muestra su preferencia en el contexto de una elección, ya sea política, social o corporativa. En el ámbito de las elecciones políticas, un voto permite a los ciudadanos elegir a sus representantes y tomar decisiones sobre leyes, políticas y directrices gubernamentales. Es un componente esencial de las democracias modernas, ya que garantiza la participación ciudadana y la legitimidad del gobierno.

Un sistema de votación es el conjunto de reglas y procedimientos que se utilizan para llevar a cabo una elección. Este sistema define cómo los votos son emitidos, recolectados, contabilizados y finalmente traducidos en resultados electorales. Los sistemas de votación son fundamentales para asegurar que las elecciones sean justas y transparentes.

Las primeras formas de votación se remontan a las civilizaciones antiguas. En la Antigua Grecia, se practicaba una forma temprana de democracia directa donde los ciudadanos varones votaban en la asamblea para tomar decisiones políticas. En un principio, el voto se expresaba mediante un método rudimentario: los ciudadanos levantaban la mano para expresar su voto. En la asamblea espartana, el voto se expresaba a través de gritos, método que hacía imposible el conteo. Aristóteles consideró esto como infantil, dado que no había una forma adecuada de contabilizar el volumen de los gritos [Rho81].

No está claro cuándo se comprendió por primera vez que el secreto era a veces deseable y podía obtenerse mediante un método adecuado de votación. Cuando se utilizaron por primera vez las papeletas para votar, el propósito no era el secreto del voto, sino realizar un conteo preciso. Por ello, empezaron a utilizar vasijas para depositar piedras en cada una de ellas, según el voto emitido. Sin embargo, en

algunas situaciones, las vasijas estaban separadas entre sí, lo que hacía evidente para el resto de las personas presentes cuál era el voto emitido.

En la actualidad, existen varios sistemas de votación que se utilizan en todo el mundo, cada uno con sus propias características y ventajas. Los más utilizados son los siguientes:

- **Votación presencial:** es el método más tradicional y común. Los votantes acuden a un centro de votación habilitado, donde es necesario que proporcionen una forma de identificación válida y se les verifica su identidad. Su voto es emitido a través de una papeleta que depositarán en una urna. Este sistema es valorado por su transparencia y la supervisión directa de los procesos electorales, pero requiere que los votantes se desplacen y supone un costo asociado con la logística y el personal.
- **Votación por correo:** permite a los votantes recibir sus papeletas en casa, marcarlas y enviarlas de vuelta a las autoridades electorales. Este método es particularmente útil para aquellos que no pueden acudir a los centros de votación. Sin embargo, plantea desafíos en términos de seguridad, debido al riesgo de papeletas extraviadas o manipuladas, así como posibles problemas de verificación de identidad. El voto por correo suele generar polémica e inseguridad entre los ciudadanos y está bajo escrutinio después de las elecciones.
- **Votación electrónica:** implica el uso de dispositivos electrónicos para emitir y registrar votos. Puede realizarse tanto en centros de votación con máquinas específicas como a través de Internet desde cualquier ubicación. Aunque este método promete mayor eficiencia y accesibilidad, también enfrenta preocupaciones significativas sobre la seguridad y la posibilidad de manipulaciones. El gobierno belga fue pionero en la aplicación de sistemas de voto electrónico comenzando en 1989. Actualmente está implantado en Bélgica, Brasil, Bulgaria, EE.UU., Emiratos Árabes Unidos, Estonia, Filipinas, India, Paraguay y Venezuela [24p].

2.1.2 Sistemas de votación basados en blockchain

En los últimos años, la votación basada en blockchain ha ganado considerable atención como una solución potencial para mejorar la transparencia, seguridad y eficiencia de los procesos electorales. Diversos estudios y proyectos han explorado y desarrollado sistemas que aplican esta tecnología para abordar las limitaciones de los métodos de votación tradicionales [JAS21].

Uno de los pilares de la votación en blockchain es su capacidad para proporcionar un registro inmutable de cada voto, lo cual es esencial para garantizar la integridad del proceso electoral. Diversos trabajos, como el de Guy Zyskind [ZNP15], han demostrado cómo una blockchain puede crear un sistema a prueba de manipulaciones, donde no es necesario confiar en terceros los datos personales ni comprometer la seguridad. Además, la naturaleza descentralizada de la blockchain permite que cualquier participante pueda verificar los resultados, lo que aumenta significativamente la confianza pública en los resultados electorales.

En 2018, Sierra Leona llevó a cabo un hito en el uso de blockchain para supervisar las elecciones generales del país [18c]. La Fundación Agora actuó como observador autorizado en las elecciones y utilizó tecnología blockchain para mejorar la transparencia y la confianza en el proceso electoral [24a]. El sistema permitió que los resultados de los votos fueran digitalizados y registrados en una blockchain, creando un registro inmutable y verificable de los resultados. Los resultados preliminares de las papeletas de votación eran subidos a la blockchain, donde se almacenaban de manera segura. El uso de blockchain en las elecciones de Sierra Leona demostró cómo esta tecnología puede aumentar la transparencia y confianza en los procesos electorales. A pesar de que no se trataba del recuento oficial [18d], la solución proporcionó un método accesible para verificar los resultados y mejorar la supervisión del proceso electoral, reduciendo así el riesgo de fraude y manipulación.

En 2018, la provincia surcoreana de Gyeonggi-do implementó un sistema de votación basado en blockchain para sus proyectos comunitarios, marcando un avance significativo en la aplicación de esta tecnología en procesos de toma de decisiones cívicas. El sistema de votación en línea, respaldado por Blocko [24b], utilizó la tecnología de los smart contracts para facilitar el proceso de votación. Blocko afirma que la tecnología, desarrollada internamente, ayudó a registrar una gran cantidad de información, incluyendo datos de los votantes, contenido de las votaciones y más, en un proceso de votación complicado, sin necesidad de supervisión o gestión por una autoridad central [17].

Otra ventaja de los sistemas de votación basados en blockchain son los incentivos para aumentar la participación electoral. Las criptomonedas y tokens pueden ser utilizados para recompensar a los votantes por su participación, lo que podría mejorar significativamente la participación ciudadana. Estos sistemas no solo motivan a los votantes, sino que también pueden integrar funcionalidades adicionales, como la distribución de recompensas por validar transacciones, fomentando así un ecosistema más activo y comprometido.

2.1.3 Análisis de la tecnología blockchain

Blockchain es una tecnología de registro distribuido que permite la transferencia segura y transparente de datos en una red de ordenadores. Funciona como un libro mayor digital, donde las transacciones se agrupan en bloques, cada uno vinculado al anterior mediante criptografía, formando una cadena. Este diseño asegura que, una vez registrados, los datos no puedan ser modificados sin alterar todos los bloques posteriores, lo que garantiza la integridad y seguridad de la información. La definición original de blockchain fue presentada por primera vez en el documento técnico de Bitcoin, publicado en 2008 por Satoshi Nakamoto, titulado "Bitcoin: A Peer-to-Peer Electronic Cash System" [Nak09].

Existen diferentes tipos de blockchain dependiendo de los datos gestionados, de la disponibilidad de dichos datos y de las acciones que un usuario puede realizar. Así, se puede distinguir entre blockchain públicas y privadas, y entre blockchain con y sin permisos [FF18].

Una blockchain pública es una red abierta y descentralizada a la que cualquiera puede unirse y participar, sin necesidad de aprobación por terceros.

- Las principales ventajas son:
 - Todas las transacciones y contratos inteligentes son visibles públicamente en la blockchain, lo que asegura la transparencia.
 - No hay una entidad central que controle la red.
- Entre los inconvenientes destaca:
 - La necesidad de que todos los nodos validen cada transacción puede llevar a problemas de escalabilidad.
 - La visibilidad pública de todas las transacciones puede comprometer la privacidad de los usuarios.

Un ejemplo de blockchain pública es Bitcoin [Nak08], la primera y más conocida. Ethereum [24d] es otro ejemplo de blockchain pública.

Una blockchain privada es una red cerrada en la que los participantes necesitan permiso para unirse. Está controlada por una sola organización o un grupo limitado de entidades.

- Las ventajas más destacadas son:

- El acceso restringido permite un mayor control sobre quién participa en la red.
- Mayor rapidez y eficiencia debido a la menor cantidad de nodos y la falta de necesidad de un mecanismo de consenso tan intensivo.
- Por otro lado, los inconvenientes principales incluyen:
 - La gestión centralizada puede introducir un punto de fallo único y depender de la confianza en la entidad que controla la red.
 - La falta de visibilidad pública puede reducir la transparencia y la capacidad de verificación de las transacciones.

La red blockchain privada más conocida es Hyperledger Fabric [Fou15], una plataforma de blockchain empresarial desarrollada por la Fundación Linux, diseñada específicamente para entornos empresariales.

Las blockchain permissionadas pueden ser públicas o privadas, pero tienen controles adicionales para determinar qué usuarios pueden realizar ciertas acciones. Ripple [Rip12], por ejemplo, utiliza un sistema de validación de transacciones con permisos donde solo entidades autorizadas pueden validar transacciones. Corda [R317], es otra blockchain con permisos diseñada para instituciones financieras.

En las blockchain sin permisos, cualquier usuario puede unirse y realizar todas las acciones sin la necesidad de una aprobación específica. Bitcoin y Ethereum son ejemplos de blockchains sin permisos.

Se ha tomado la decisión de utilizar Ethereum, una blockchain pública y sin permisos, para el desarrollo de este TFM. A continuación se listan los motivos:

- Todas las transacciones y contratos inteligentes son visibles públicamente en la blockchain, lo que asegura la transparencia.
- Ethereum es descentralizado, por lo que no hay una entidad central que controle la red.
- No se requiere permiso para unirse a la red y participar en las actividades de la blockchain.
- Ethereum permite la creación y ejecución de smart contracts por cualquier usuario de la red.

2.1.4 Análisis sobre las mejoras de seguridad que se pueden implementar en el sistema

Existen diversas tecnologías que siguen los principios de la descentralización sobre los cuales se basa blockchain y que pueden ser implementadas en sistemas de votación electrónica para mejorar significativamente su seguridad. Estas tecnologías no solo complementan a la blockchain, sino que también refuerzan la integridad, disponibilidad y confiabilidad del sistema:

- **IPFS (InterPlanetary File System):** un protocolo y red peer-to-peer para almacenar y compartir datos en un sistema de archivos distribuido. La implementación de IPFS en el sistema puede mejorar la disponibilidad al distribuir los datos en múltiples nodos, asegurando que los datos almacenados sean inmutables mediante la identificación de cada archivo con un hash criptográfico. Esto garantiza que los votos y otros datos electorales no puedan ser alterados sin ser detectados. Además, la naturaleza descentralizada de IPFS garantiza que los datos permanezcan accesibles incluso si algunos nodos fallan.
- **Orbit DB:** una base de datos distribuida y descentralizada construida sobre IPFS y utilizada comúnmente con blockchain. Su integración en sistemas de votación puede ofrecer varias ventajas de seguridad ya que, al descentralizar el almacenamiento de datos, OrbitDB elimina la necesidad de un servidor central, reduciendo el riesgo de ataques y manipulaciones.
- **Oráculos:** Son servicios que permiten que los contratos inteligentes en la blockchain interactúen con datos externos a la blockchain. Su uso en los sistemas de votación facilita que los contratos inteligentes (smart contracts) accedan y utilicen datos que se encuentran fuera de la blockchain.
- **Zero-Knowledge Proofs (ZKPs):** Esta tecnología permite verificar la autenticidad de una transacción sin revelar información confidencial, garantizando la privacidad del votante mientras se asegura la integridad del voto.

2.2 Tecnologías base

2.2.1 Ethereum

Ethereum [24d] es una plataforma descentralizada de código abierto que permite a los desarrolladores construir y desplegar aplicaciones descentralizadas (DApps)

utilizando tecnología blockchain. Ethereum es una blockchain pública y sin permisos. Su característica distintiva es la capacidad de ejecutar contratos inteligentes o smart contracts, que son programas informáticos autónomos diseñados para ejecutar automáticamente y hacer cumplir acuerdos digitales de manera transparente y confiable sin necesidad de intermediarios.

Ethereum ha sido una plataforma pionera en la implementación de contratos inteligentes, lo que la convierte en un candidato ideal para sistemas de votación electrónica. A lo largo de los años, se han desarrollado diversos modelos y proyectos piloto que utilizan la blockchain de Ethereum para procesos electorales.

2.2.2 Node.js

Node.js [24j] es un entorno de ejecución de JavaScript basado en el motor V8 de Google Chrome. Aunque inicialmente se utilizaba principalmente en el desarrollo del lado del servidor, Node.js ha evolucionado para convertirse en una herramienta clave en el desarrollo de aplicaciones descentralizadas (DApps). Node.js permite a los desarrolladores utilizar JavaScript tanto en el lado del cliente como en el lado del servidor. Esto proporciona coherencia en el código y permite a los desarrolladores trabajar en el mismo lenguaje en todos los aspectos de la aplicación. Ofrece bibliotecas y frameworks que facilitan la conexión de la aplicación con la blockchain. Esto incluye bibliotecas como Web3.js, que permite interactuar con contratos inteligentes y realizar transacciones en la blockchain. Node.js utiliza npm (Node Package Manager), que incluye herramientas como npm audit para identificar y solucionar vulnerabilidades en las dependencias del proyecto. Esta capacidad de auditoría automatizada ayuda a mantener el proyecto seguro al detectar y alertar sobre problemas conocidos en las bibliotecas de terceros utilizadas. El modelo de E/S no bloqueante de Node.js es particularmente útil para manejar múltiples solicitudes y conexiones simultáneamente, lo que es esencial para las DApps que interactúan constantemente con la blockchain. Este modelo también puede ayudar a mitigar ciertos tipos de ataques, como el DoS (Denial of Service), ya que puede manejar una gran cantidad de conexiones sin quedar bloqueado.

2.2.3 Solidity

Solidity [24o] es un lenguaje de programación diseñado específicamente para el desarrollo de contratos inteligentes en Ethereum. La razón principal por la que Solidity se utiliza en el contexto de las aplicaciones descentralizadas (DApps) es su

capacidad para escribir y desplegar contratos inteligentes de manera eficiente en la blockchain. Solidity está diseñado con características de seguridad incorporadas para ayudar a los desarrolladores a escribir contratos inteligentes seguros y resistentes a errores. Por ejemplo, incluye soporte para varios tipos de datos complejos, control de acceso, y mecanismos de verificación formal. Además, Solidity proporciona herramientas para manejar de manera efectiva situaciones comunes en contratos inteligentes, como la transferencia de tokens, la creación de estructuras de datos, y la gestión de excepciones.

2.2.4 MetaMask

MetaMask [24i] es una extensión de navegador y una billetera digital que permite a los usuarios interactuar con aplicaciones descentralizadas (DApps) en la blockchain de Ethereum. Está diseñado con una interfaz intuitiva y fácil de usar que permite a los usuarios realizar transacciones en la blockchain de Ethereum con solo unos pocos clics. MetaMask destaca por su enfoque en la seguridad y la privacidad, implementando medidas como la encriptación de claves privadas y la aprobación manual de cada transacción. También permite la conexión segura con múltiples DApps sin necesidad de descargar nodos completos de la blockchain, facilitando así la interacción con contratos inteligentes y servicios descentralizados. MetaMask también incluye un sistema de gestión de identidades, lo que permite a los usuarios controlar sus claves privadas y realizar transacciones de manera segura y privada.

2.2.5 Hardhat

Hardhat [24f] es un framework de desarrollo de Ethereum que proporciona un conjunto de herramientas y utilidades para facilitar la creación, compilación, prueba y despliegue de aplicaciones descentralizadas (DApps) y contratos inteligentes en la blockchain de Ethereum. Es una de las opciones más populares entre los desarrolladores de Ethereum debido a su potencia, flexibilidad y facilidad de uso.

Hardhat permite a los desarrolladores configurar una blockchain local privada para pruebas. Esta blockchain local, conocida como Hardhat Network, simula una red pública como Ethereum pero se ejecuta en el entorno de desarrollo del desarrollador.

Hardhat, por defecto, utiliza un algoritmo de consenso de prueba de trabajo (PoW, Proof of Work). PoW es un mecanismo de consenso en el que los participantes de

una red blockchain, conocidos como mineros, compiten para resolver un problema matemático complicado. El primer minero en resolver el problema valida el bloque de transacciones y recibe una recompensa en forma de criptomoneda.

2.2.6 IPFS

IPFS (InterPlanetary File System) [24h] es un protocolo de red peer-to-peer que permite la creación de un sistema de archivos distribuido y descentralizado. Desarrollado por Protocol Labs, IPFS se basa en la idea de un sistema de archivos que conecta todos los dispositivos informáticos con el mismo sistema de archivos, de forma similar a cómo el protocolo HTTP conecta todos los dispositivos a través de la web. A diferencia de HTTP, que se basa en la ubicación para encontrar archivos, IPFS se basa en el contenido. Esto significa que cada archivo y todos los bloques dentro de él son direccionables a través de un hash criptográfico, proporcionando una manera más eficiente y segura de almacenar y compartir datos. Además, como los archivos se identifican por su contenido en lugar de su ubicación, IPFS permite la duplicación de datos y el almacenamiento eficiente, ya que múltiples copias del mismo archivo pueden ser referenciadas por un solo hash.

Al ser descentralizado, IPFS elimina la dependencia de servidores centralizados, lo que reduce la vulnerabilidad a ataques de denegación del servicio y fallos del servidor.

2.2.7 OrbitDB

OrbitDB [24l] es una base de datos descentralizada y distribuida que funciona sobre IPFS. Diseñada para aplicaciones peer-to-peer, OrbitDB permite a los desarrolladores crear bases de datos sin necesidad de servidores centralizados, lo que asegura una mayor disponibilidad de datos. Además, OrbitDB incluye mecanismos de replicación y sincronización automáticos, facilitando la consistencia de los datos entre múltiples nodos en la red.

Ofrece varios tipos de bases de datos, como logs (registros), key-value stores (almacenes clave-valor), índices, y bases de datos de documentos. Cada entrada en la base de datos es identificada por un hash criptográfico, lo que permite la verificación de datos y la protección contra manipulaciones.

Diseño del sistema

3.1 Definición de los requisitos

El ámbito de los procesos electorales es sumamente diverso, con una amplia gama de enfoques y métodos utilizados en todo el mundo. Desde sistemas de voto presencial tradicionales hasta votaciones electrónicas complejas, cada proceso electoral tiene sus propias características y desafíos únicos.

En el contexto de este trabajo, nos enfocaremos exclusivamente en la seguridad del proceso electoral, dejando de lado consideraciones sobre la complejidad de los métodos de votación utilizados. Por lo tanto, nuestro enfoque se centrará en garantizar la integridad, seguridad y transparencia del sistema de votación, independientemente del tipo específico de votación que se utilice.

Dicho esto, para simplificar el sistema de votación y centrarnos en la seguridad, vamos a limitar nuestras consideraciones a un escenario básico donde los votantes simplemente eligen entre diferentes candidatos sin tener en cuenta aspectos más complejos como la división de votos por regiones geográficas o provincias. No obstante, es importante destacar que el código será diseñado con flexibilidad, de manera que pueda adaptarse y ser modificado en el futuro para incorporar otros tipos de votaciones o funcionalidades adicionales.

3.1.1 Requisitos no funcionales

- **Seguridad y autenticación:** El sistema debe garantizar la seguridad y la autenticación de los votantes, evitando la duplicación de votos y asegurando que solo los votantes autorizados puedan emitir votos.
- **Integridad de los votos:** Los votos emitidos deben registrarse de manera segura y no deben ser modificables ni manipulables una vez emitidos.
- **Transparencia y verificabilidad:** El sistema debe ser transparente, permitiendo a todos los participantes verificar la validez de los votos y el proceso de votación en su conjunto.

- **Privacidad de los votantes:** Se debe garantizar la privacidad de los votantes, protegiendo la identidad y la elección de cada votante.
- **Escalabilidad:** El sistema debe ser capaz de manejar un gran número de votantes y votaciones simultáneas de manera eficiente.

3.1.2 Requisitos funcionales

Dentro del sistema distinguiremos dos roles, el rol del votante y el rol del administrador:

Votantes:

Son los individuos que participan en el proceso de votación. Deben tener acceso al sistema para emitir sus votos de manera segura. Deben poder verificar que su voto ha sido registrado correctamente y que no ha sido alterado. El proceso que lleva a cabo un votante se resume en la Fig. 3.1 y consta de los siguientes pasos principales:

- **Autenticarse:** El votante debe autenticarse para acceder al sistema de votación y garantizar que solo los votantes autorizados puedan emitir su voto. Por ejemplo, en el sistema presentado en este TFM, los usuarios se autentican en el sistema mediante la firma de un mensaje generado aleatoriamente utilizando MetaMask u otro proveedor de billetera Ethereum. Se verifica la autenticidad de la firma y se comprueba si la cuenta que ha firmado el mensaje corresponde a una de las cuentas registradas por los administradores en el sistema de votación.
- **Seleccionar votación:** Una vez autenticado, al votante se le presenta una lista de elecciones disponibles y seleccionará la elección en la que desee participar.
- **Votar:** Después de seleccionar la elección, al votante se le muestran las opciones de voto disponibles, como los candidatos o las propuestas. El votante seleccionará las opciones de voto y enviará su voto al sistema. Una vez emitido el voto, este se almacenará en una base de datos descentralizada (por ejemplo, en el sistema presentado en este TFM, se utiliza una base de datos basada en IPFS).

- Comprobar los datos de sus votos: Una vez emitido el voto, el votante puede verificar los datos de sus votos emitidos para garantizar la exactitud y la integridad del proceso.
- Obtener resultados: El administrador obtiene los resultados de la votación en tiempo real.

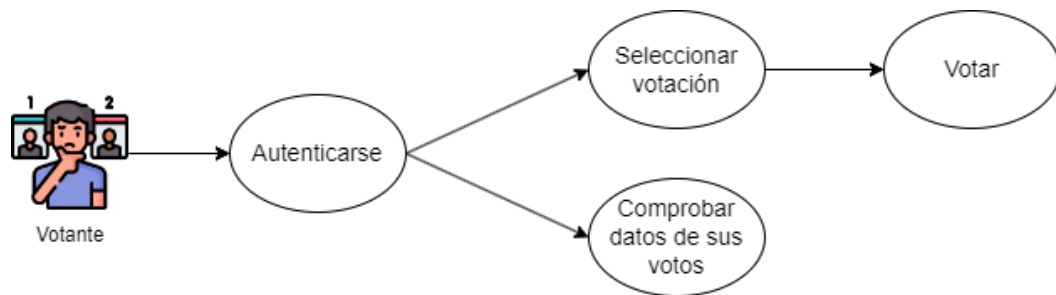


Figura 3.1.: Requisitos funcionales del votante.

Administradores:

Los administradores representan instituciones o empresas encargadas de desplegar los contratos inteligentes relacionados con las votaciones y de completar todos los datos necesarios para ellas. Son responsables de configurar y gestionar las elecciones dentro del sistema. Tienen acceso a herramientas administrativas para definir los parámetros de la votación, como las opciones de voto, las fechas de inicio y fin, etc. Pueden realizar las acciones ilustradas en la Fig. 3.2, que incluyen la posibilidad de:

- Registrar votantes: El administrador registra a los votantes elegibles en el sistema para permitirles participar en la elección.
- Crear votación: El administrador crea una votación, introduciendo los datos requeridos para votar.
- Cerrar votación: El administrador cierra el proceso de votación, impidiendo que los votantes emitan sus votos una vez que la votación está cerrada.
- Obtener resultados: El administrador obtiene los resultados de la votación en tiempo real.

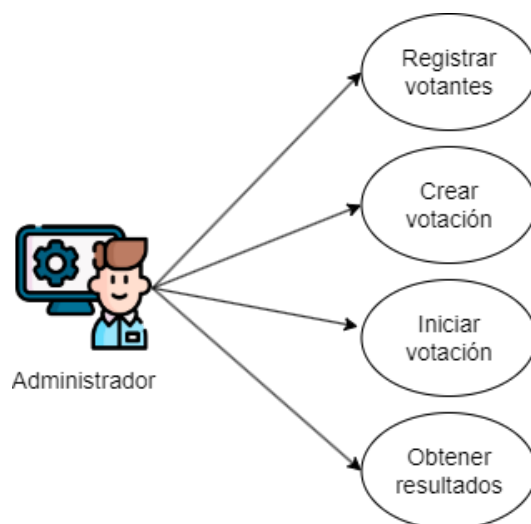


Figura 3.2.: Requisitos funcionales del administrador.

3.1.3 Casos de uso

Registro

El registro de los votantes en el sistema es una de las partes más importantes en términos de seguridad. La dificultad de este proceso radica en la necesidad de asociar cada una de las cuentas Ethereum utilizadas con la identidad de un votante autorizado para usar el sistema de votación. Si este proceso falla, se pone en peligro la privacidad de los votantes al relacionar un voto emitido con la identidad de uno de ellos.

En España, la protección de los datos personales y la privacidad en el contexto de un sistema de votación se rige principalmente por el Reglamento General de Protección de Datos (RGPD) [16] y la Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales (LOPDGDD) [18b]. Además, en el contexto de una elección gubernamental, tenemos que tener en cuenta la Ley Orgánica del Régimen Electoral General (LOREG) [85], que en el artículo 86 garantiza el derecho al voto secreto, lo que implica que no se debe asociar la identidad del votante con su voto emitido para proteger su privacidad y evitar cualquier tipo de coacción o represalia.

Si asociamos el voto con la identidad del votante en un sistema de votación en España, podríamos violar varias leyes y principios fundamentales de protección de datos y privacidad, lo que resultaría en graves consecuencias legales.

En nuestro sistema, la fase de registro de votantes será administrada por los administradores. Al desplegar el sistema, estos deben definir una lista de votantes autorizados. En nuestro enfoque, se generará una cuenta de Ethereum única para cada votante autorizado. De este modo, los responsables de desplegar el sistema también serán los encargados de generar las cuentas de Ethereum para cada votante.

El método para registrar a los votantes variará en función de la red blockchain que se utilice y los mecanismos específicos disponibles para la creación de cuentas en cada entorno. En este proyecto, hemos utilizado Hardhat como red de pruebas, lo cual ha influido en nuestra aproximación al registro de usuarios.

Hardhat crea automáticamente varias cuentas con sus respectivas llaves privadas al iniciar un entorno de desarrollo. Estas cuentas preconfiguradas están listas para ser utilizadas inmediatamente en las pruebas, eliminando la necesidad de implementar un proceso manual o automatizado de registro de votantes. Debido a esta funcionalidad automática, no hemos considerado necesario implementar un proceso detallado de registro de usuarios dentro del sistema. La facilidad de uso y la automatización proporcionada por Hardhat nos permiten concentrarnos en otras áreas críticas del desarrollo y pruebas de nuestro sistema de votación.

Si se decidiera utilizar una red diferente, como una basada en Geth (Go Ethereum), el proceso de creación y registro de cuentas requeriría un enfoque más elaborado. Geth es un cliente de Ethereum que permite interactuar con la red de Ethereum, y su configuración para el manejo de cuentas es diferente de la de Hardhat. En una red basada en Geth, el registro de votantes podría automatizarse utilizando scripts que gestionen la creación de cuentas y su inclusión en el sistema de votación.

Autenticación

Al usar MetaMask y gracias a la decisión de que los administradores lleven a cabo el proceso de registro, se consigue simplificar el proceso de autenticación de los usuarios. Los usuarios se conectarán a la aplicación utilizando MetaMask, como se puede ver en la Fig. 3.3, y estarán de esta forma autenticados en todo momento.

Seleccionar votación

En este proceso, el votante comienza enviando una solicitud a la aplicación para obtener la lista de votaciones abiertas. La aplicación toma esta solicitud y la envía

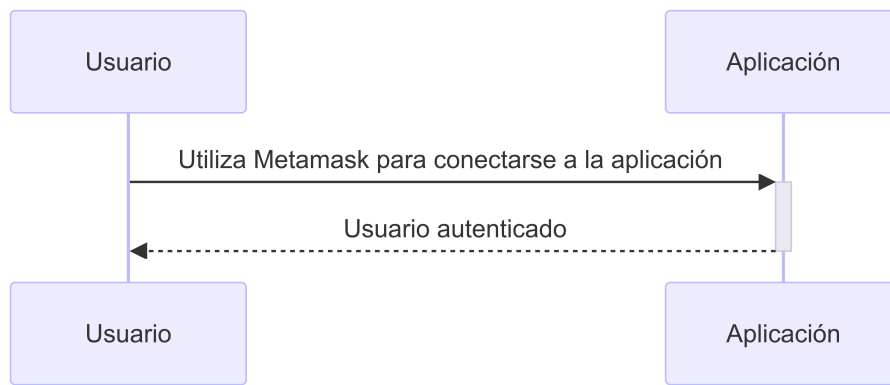


Figura 3.3.: Diagrama de secuencia del proceso de autenticación.

al nodo OrbitDB, que es responsable de almacenar la información de las votaciones. El nodo OrbitDB procesa la solicitud y devuelve la lista de votaciones abiertas a la aplicación, indicando el título y la dirección del smart contract de cada una de las votaciones.

Una vez que la aplicación recibe la lista de votaciones abiertas del nodo OrbitDB, la muestra al votante. El votante entonces examina las opciones y selecciona una votación específica de la lista proporcionada. La aplicación, al recibir la selección del votante, realiza una llamada a un smart contract en la blockchain correspondiente a la votación seleccionada. Este smart contract en la blockchain procesa la solicitud y devuelve las opciones disponibles para esa votación a la aplicación. Finalmente, la aplicación recibe las opciones de votación desde la blockchain y las muestra al votante, permitiéndole ver las alternativas disponibles en la votación seleccionada. Este proceso se muestra en la Fig. 3.4.

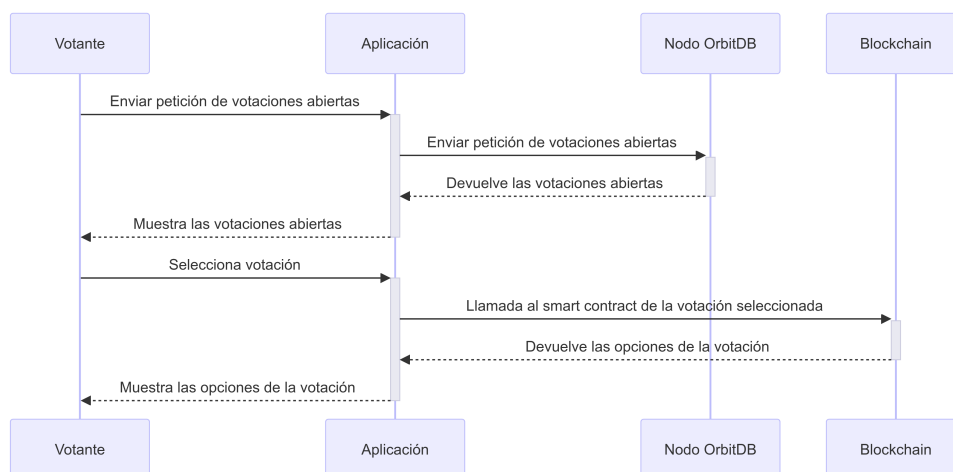


Figura 3.4.: Diagrama de secuencia del proceso de seleccionar votación.

Votar

Al momento de votar, el votante utilizará la aplicación para seleccionar una opción de voto y enviará una transacción a la red blockchain a través de Metamask. Esta transacción también emitirá un evento que incluirá los datos del voto emitido. Mientras tanto, el oráculo, que monitorea la red blockchain, escuchará el evento emitido y almacenará el voto en IPFS. La blockchain emitirá una confirmación de la transacción, que la aplicación recogerá y mostrará al usuario, indicándole que su voto ha sido registrado exitosamente. Este proceso se muestra en la Fig. 3.5.

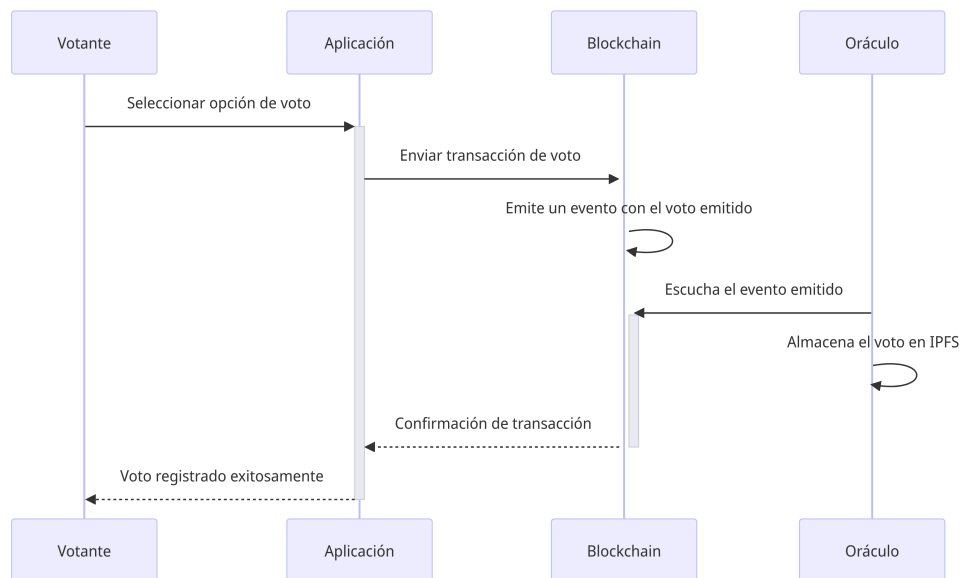


Figura 3.5.: Diagrama de secuencia del proceso de votar.

Comprobar los datos de sus votos

Cuando el votante quiera comprobar los datos de su voto, la aplicación, respondiendo a esta solicitud, enviará una petición a la blockchain para comprobar el voto. Una vez que la blockchain recibe esta petición, emite un evento asociado con la solicitud de verificación del voto.

El oráculo, que está monitorizando la blockchain, detecta el evento emitido y actúa en consecuencia. El oráculo accede a IPFS para recuperar el voto almacenado y lo devuelve a la blockchain. La blockchain, con los datos del voto recuperados, responde a la aplicación. Finalmente, la aplicación muestra al votante los datos de su voto. Este proceso se muestra en la Fig. 3.6.

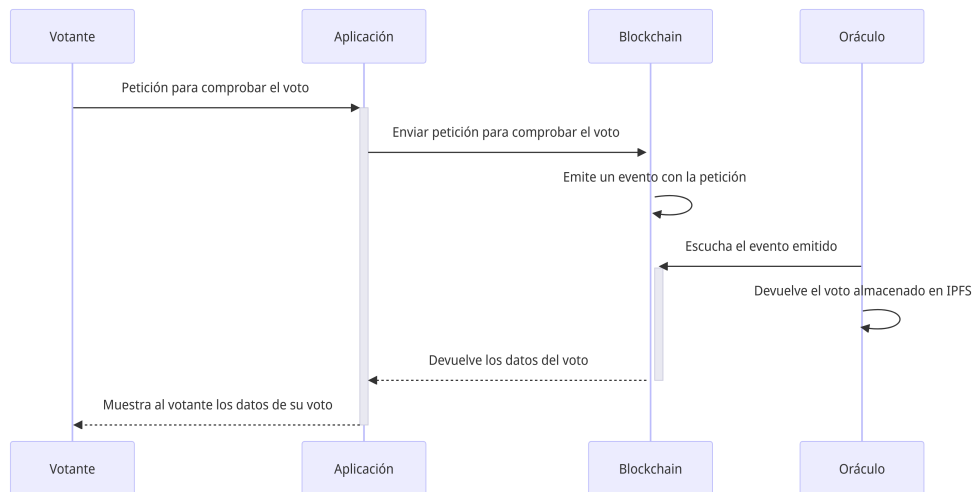


Figura 3.6.: Diagrama de secuencia del proceso de comprobar el voto.

Crear votación

Tal y como se muestra en la Fig. 3.7, en este proceso, el administrador comienza introduciendo los datos de la votación en la aplicación. La aplicación toma estos datos y los utiliza para desplegar un smart contract en la blockchain utilizando MetaMask. Una vez que el smart contract ha sido desplegado, la blockchain envía una confirmación a la aplicación. La aplicación recibe esta confirmación y muestra el resultado del despliegue al administrador, informándole que el smart contract ha sido desplegado correctamente.

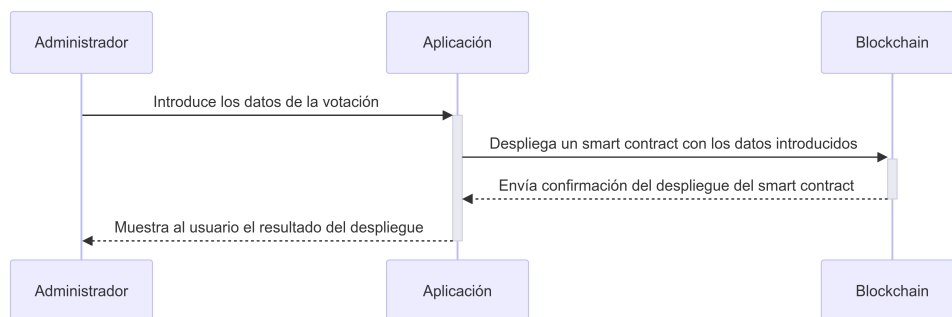


Figura 3.7.: Diagrama de secuencia del proceso de crear una votación.

Cerrar votación

El administrador interactúa con la aplicación para cerrar una votación. La aplicación, a su vez, comunica esta solicitud al smart contract en la blockchain, indicando

que la votación debe cerrarse. El smart contract procesa esta solicitud y se cierra. Una vez que el smart contract está cerrado, la blockchain envía una confirmación de cierre a la aplicación. La aplicación, al recibir esta confirmación, muestra al administrador que la votación ha sido cerrada exitosamente. Este proceso se muestra en la Fig. 3.8.

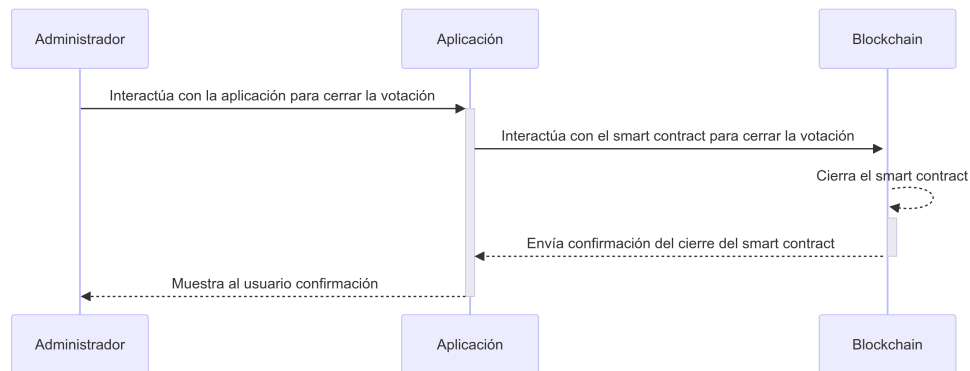


Figura 3.8.: Diagrama de secuencia del proceso de iniciar una votación.

Obtener resultados

Tal y como se muestra en la Fig. 3.9, el administrador o votante interactúa con la aplicación para enviar una petición de recuento de votos al smart contract. El smart contract realizará el recuento de los votos y enviará ese recuento a la aplicación, que luego lo mostrará al administrador o votante.

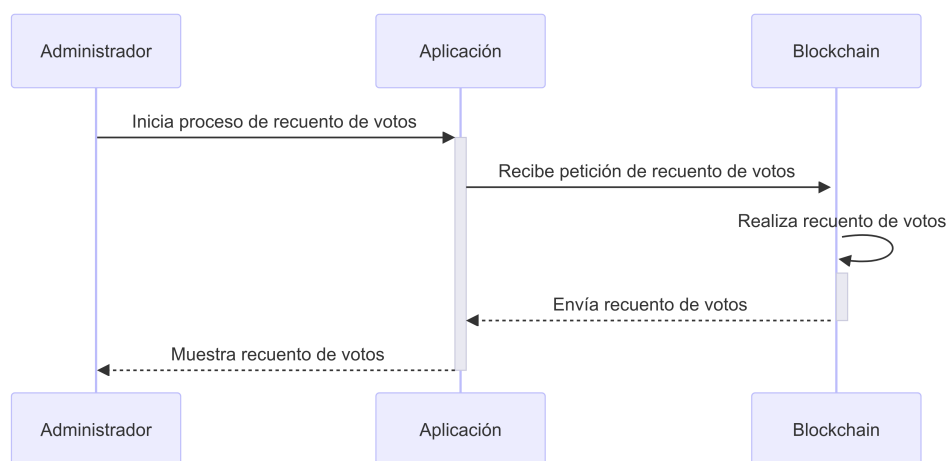


Figura 3.9.: Diagrama de secuencia del proceso del recuento de votos.

3.2 Arquitectura del sistema

La arquitectura del sistema propuesto se muestra en la Fig. 3.10. En ella se puede ver que los usuarios interactúan con el sistema a través de un front-end basado en Node.js. Este front-end permitirá a los usuarios realizar diversas operaciones utilizando una cartera digital como MetaMask, la cual les facilitará conectarse a su cuenta de Ethereum y, de esta forma, interactuar con los contratos inteligentes desplegados en la blockchain.

Cada votación dentro del sistema tendrá su propio contrato inteligente asociado. Este contrato será desplegado por el administrador del sistema, quien será responsable de configurar los parámetros y condiciones de cada votación. Una vez que el contrato esté desplegado, los usuarios podrán interactuar con él a través del front-end, utilizando MetaMask para autenticar y firmar sus transacciones, y así ejercer su voto de manera segura y transparente.

Para gestionar la información de los contratos inteligentes de manera eficiente, se utilizará almacenamiento descentralizado mediante OrbitDB. OrbitDB permitirá almacenar de manera distribuida las direcciones de cada uno de los contratos inteligentes desplegados, asegurando que la información esté accesible y no dependa de un solo punto de fallo, lo que aumenta la robustez y la disponibilidad del sistema.

Cada vez que un usuario emita su voto, el sistema realizará una llamada a un oráculo descentralizado. Este oráculo se encargará de subir a la red IPFS un archivo que contendrá los datos del voto emitido por el usuario. Antes de subir el archivo a la red IPFS, se cifrará utilizando una contraseña. Este cifrado garantiza que, aunque el archivo esté disponible públicamente en la red IPFS, solo aquellas personas que posean la contraseña podrán descifrar y leer su contenido. Este archivo cifrado actuará como un recibo que prueba el voto emitido por el usuario, proporcionando una capa adicional de seguridad y verificabilidad.

El uso de una red IPFS para almacenar los archivos de los votos asegura que estos datos sean inmutables y distribuidos, eliminando el riesgo de manipulación y garantizando la integridad del proceso de votación. Además, el cifrado de los archivos protege la privacidad de los votantes, asegurando que la información sensible solo pueda ser accedida por quienes están autorizados.

Utilizar un almacenamiento descentralizado, como una base de datos OrbitDB, presenta varias ventajas significativas en comparación con las soluciones de almacenamiento centralizadas. En un sistema descentralizado, los datos se replican a través de múltiples nodos, lo que significa que si uno o varios nodos fallan, los datos aún

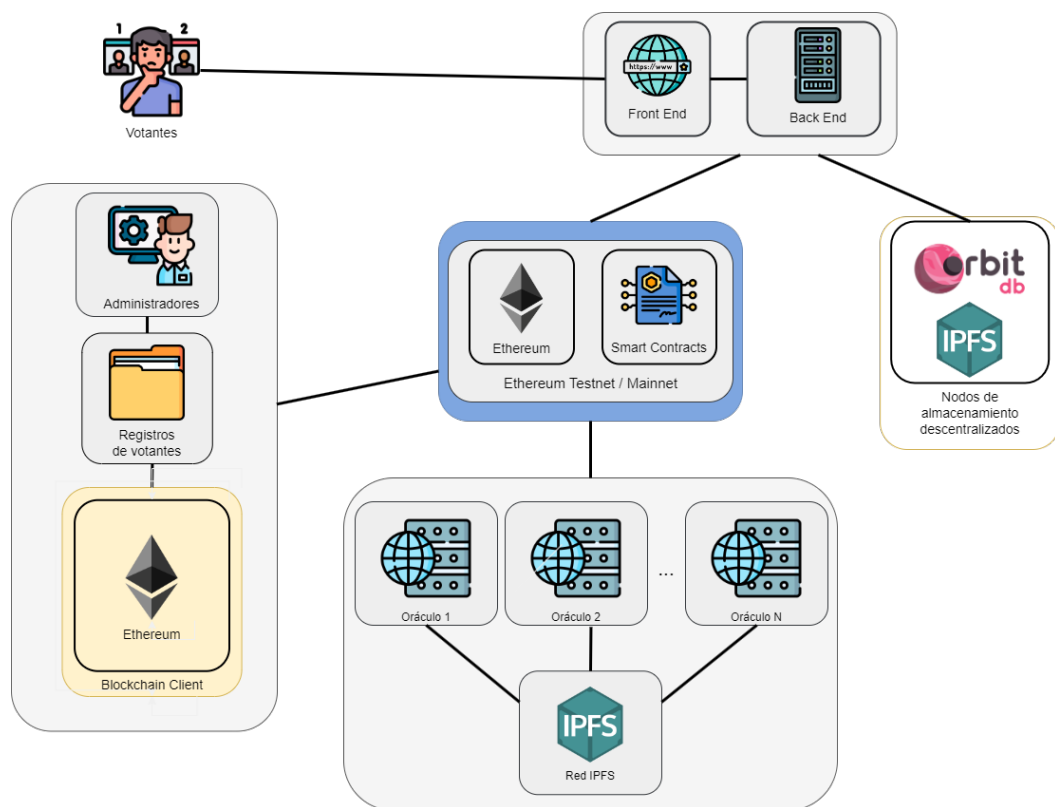


Figura 3.10.: Arquitectura del sistema.

estarán disponibles en otros nodos. Esto reduce el riesgo de pérdida de datos y garantiza una alta disponibilidad. Además, las bases de datos descentralizadas pueden escalar de manera más eficiente que las centralizadas. A medida que se añaden más nodos a la red, la capacidad de almacenamiento y procesamiento aumenta sin depender de un servidor central.

Otra ventaja clave del almacenamiento descentralizado con OrbitDB es la mejora en la seguridad. En una base de datos centralizada, un único punto de fallo puede ser explotado por atacantes, comprometiendo toda la integridad del sistema. Sin embargo, en un sistema descentralizado, los datos se distribuyen entre múltiples nodos, lo que dificulta significativamente los intentos de ataque. Los nodos pueden implementar medidas de consenso y validación que aseguran que cualquier modificación o adición a la base de datos sea verificada y aceptada por la mayoría de los nodos, lo que fortalece la resistencia a manipulaciones maliciosas.

La transparencia es otra ventaja significativa de utilizar OrbitDB. Las operaciones y transacciones en una base de datos descentralizada pueden ser más transparentes y auditables. La capacidad de auditar públicamente las transacciones garantiza que todas las partes involucradas puedan verificar la integridad y la exactitud de los datos.

Implementación del sistema

4.1 Smart Contract

Al implementar el sistema, el primer componente en ser desarrollado será el contrato inteligente (Smart Contract), ya que es la base sobre la cual se fundamenta el sistema. Es crucial seguir todas las recomendaciones de seguridad para asegurar que el contrato inteligente sea seguro y no pueda ser explotado por ningún atacante.

El contrato inteligente debe contar con una serie de funciones mínimas:

- Una función que permita al dueño del contrato añadir los campos de las diferentes opciones entre las que se puede votar.
- Una función que registre si un usuario que usa el contrato ya ha realizado un voto anteriormente. Esto evita que un mismo usuario pueda votar más de una vez, garantizando la integridad del proceso de votación.
- Una función que permita a los usuarios votar eligiendo una de las opciones definidas dentro del contrato inteligente.
- Una función que permita obtener el recuento total de votos. Esta función debe ser accesible a todo el mundo para proporcionar transparencia y permitir la verificación de los resultados de la votación.
- Una función que permita cerrar las votaciones, que únicamente estará disponible para el dueño del contrato.

Además, es importante restringir el acceso a funciones específicas únicamente al propietario del contrato (generalmente la cuenta que desplegó el contrato o una cuenta designada como propietaria). Al restringir el acceso a funciones específicas, se evita que usuarios malintencionados o inexpertos realicen cambios que podrían comprometer el contrato o su lógica. Sin restricciones adecuadas, cualquier usuario podría llamar a funciones que modifican el estado del contrato, posiblemente causando daños, intencionales o no. Funciones como cambiar parámetros importantes, retirar fondos o actualizar la lógica del contrato deben estar protegidas para evitar

vulnerabilidades. Además, al limitar el acceso a funciones sensibles a una sola entidad (el propietario), se puede atribuir la responsabilidad de las acciones realizadas en el contrato, lo que facilita la gestión y auditoría.

Para controlar que ciertas funciones solo puedan ser realizadas por el propietario en un contrato inteligente de Solidity, se utiliza comúnmente un patrón llamado "Ownable". Este patrón define un propietario del contrato y permite restringir el acceso a funciones específicas solo a esta cuenta. La biblioteca OpenZeppelin [24k] proporciona una implementación segura y bien probada de este patrón, lo que facilita la creación de contratos inteligentes seguros y robustos. Al utilizar la biblioteca OpenZeppelin en lugar de nuestra propia implementación, nos aseguramos una implementación segura, evitando los posibles errores que podríamos cometer. Un ejemplo de esta implementación se encuentra en el List. 4.1.

List. 4.1: Patrón "Ownable" de OpenZeppelin.

```
1 // Ejemplo de implementación utilizando OpenZeppelin
import "@openzeppelin/contracts/access/Ownable.sol";

contract VotingContract is Ownable {
5   // Implementación de funciones de votación...
}
```

Para prevenir los votos duplicados, el contrato inteligente registrará las direcciones que ya han votado. Este registro nos permitirá asegurar que cada dirección solo pueda emitir un voto por cada proceso de votación.

Utilizaremos un mapeo (mapping) en Solidity para almacenar las direcciones de los votantes. Este mapeo asociará cada dirección con un valor booleano que indica si la dirección ya ha votado. Antes de permitir que una dirección vote, se verificará en el mapeo si la dirección ya ha emitido un voto. Si la dirección ya está registrada como votante, se rechazará la transacción. Dado que los datos en la blockchain son inmutables, una vez que una dirección es registrada como votante, no se puede alterar, evitando cualquier intento de manipulación.

Al crear una nueva votación, se realizará una llamada al smart contract a través del constructor. El constructor recibirá tres parámetros: uno de ellos será el título, una cadena de texto (string) que permitirá a los usuarios identificar fácilmente la votación. Los otros dos parámetros serán las dos opciones disponibles entre las cuales los usuarios podrán votar. Tal como se explicó en la subsección 3.1, por simplicidad, este ejemplo ilustra la forma más sencilla de votación. Sin embargo, este código puede ser reutilizado y extendido para crear votaciones mucho más complejas con diferentes tipos de elecciones y opciones para los usuarios. Un ejemplo de esta implementación se encuentra en el List. 4.2.

List. 4.2: Constructor del smart contract.

```
1 constructor(string memory _title, string memory _option1, string memory
    _option2) {
    title = _title;
    option1 = _option1;
    option2 = _option2;
5 }
```

La función "vote" es el mecanismo principal que permite a los usuarios participar en la votación. La función recibirá un parámetro llamado "option", que es una cadena de texto (string). Este parámetro representa la opción que el usuario elige durante el proceso de votación. La función es pública, lo que significa que puede ser llamada por cualquier persona en la red de Ethereum, siempre y cuando se cumplan las condiciones establecidas por los modificadores:

- **hasNotVoted:** Este modificador asegura que el usuario que llama a la función no haya votado previamente. Si el usuario ya ha emitido un voto, el modificador impedirá que se ejecute el resto de la función.
- **votingIsOpen:** Este modificador verifica que la votación esté actualmente abierta. Si la votación está cerrada, el modificador impedirá que se ejecute el resto de la función.

La función comprobará que la opción proporcionada por el usuario coincide con alguna de las opciones válidas. Se realiza una comparación de los hashes de la opción proporcionada con los hashes de las dos opciones válidas utilizando la función `keccak256`. Si el hash de la opción no coincide con ninguno de los hashes de las opciones válidas, se emitirá un mensaje de error "Invalid option" y la función se detendrá. Una vez que la opción es validada, la dirección del votante se marca en un registro para indicar que el usuario ya ha votado. Esto se logra actualizando el mapeo `hasVoted` con el valor `true` para la dirección del votante (`msg.sender`). La función verifica cuál opción ha sido seleccionada comparando el hash de la opción proporcionada y se incrementará el contador de votos correspondiente. Finalmente, se emite un evento llamado `Voted` para registrar que un usuario ha votado. Este evento incluye la dirección del votante (`msg.sender`) y la opción elegida (`option`). Esta implementación se encuentra en el List. 4.2.

List. 4.3: Función vote.

```
1 modifier hasNotVoted() {
    require(!hasVoted[msg.sender], "You have already voted");
    _;
}
5 modifier votingIsOpen() {
    require(!votingClosed, "Voting is closed");
}
```

```

-;
}

10 function vote(string memory option) public hasNotVoted votingIsOpen {
    require(
        keccak256(abi.encodePacked(option)) == keccak256(abi.
                                                    encodePacked(option1)) |
        keccak256(abi.encodePacked(option)) == keccak256(abi.
                                                    encodePacked(option2)),
        "Invalid option"
15    );

    hasVoted[msg.sender] = true;

    if (keccak256(abi.encodePacked(option)) == keccak256(abi.
                                                            encodePacked(option1)))
20    {
        votesOption1++;
    } else {
        votesOption2++;
    }

25    emit Voted(msg.sender, option);
}

```

La función `getVotes` se encarga de proporcionar la cantidad de votos que cada opción ha recibido en la votación. La función es `public`, lo que significa que puede ser llamada por cualquier persona, tanto desde dentro del contrato como desde aplicaciones externas. El modificador `view` indica que esta función solo lee datos del estado del contrato, pero no los modifica, no realiza ninguna acción que cambie el estado de la blockchain. La función devuelve dos valores de tipo `uint256`, que son los contadores de votos para cada una de las dos opciones disponibles. Esta función se muestra en el List. 4.4.

List. 4.4: Función `getVotes`.

```

1 function getVotes() public view returns (uint256, uint256) {
    return (votesOption1, votesOption2);
}

```

La función `closeVoting` se utiliza para cerrar la votación, impidiendo que más votos sean emitidos. La función es `public`, por lo que puede ser llamada desde fuera del contrato, pero está restringida por el modificador `onlyOwner`, que asegura que solo la persona que es el propietario del contrato (generalmente la cuenta que desplegó el contrato) puede llamar a esta función. Esto impide que cualquier usuario no autorizado cierre la votación. Esta función se muestra en el List. 4.5.

La función cambia el estado de la variable `votingClosed` a `true`, indicando que la votación ha sido cerrada y emite un evento `VotingClosed`, que notifica a la blockchain y a cualquier aplicación o usuario interesado que la votación ha terminado.

List. 4.5: Función `closeVoting`.

```
1 function closeVoting() public onlyOwner {  
    votingClosed = true;  
    emit VotingClosed();  
}
```

Antes de desplegar el contrato, es necesario compilar el código fuente del smart contract para obtener dos componentes clave:

- **Bytecode:** El bytecode es el código binario que se desplegará en la blockchain. Es la forma ejecutable del contrato inteligente que será interpretado por la Ethereum Virtual Machine (EVM).
- **ABI (Application Binary Interface):** El ABI es una interfaz que define cómo interactuar con el contrato inteligente desde fuera de la blockchain. Es una estructura de datos que describe las funciones disponibles en el contrato, los tipos de datos que se pueden enviar y recibir, y los eventos que el contrato puede emitir.

Una vez obtenido el bytecode y el ABI, se utilizará MetaMask para desplegar el contrato en la red de Ethereum. A través del front-end, el administrador puede proporcionar los datos necesarios, como los parámetros del constructor del contrato, y enviar una transacción para desplegar el contrato en la blockchain.

El sistema está diseñado para permitir el despliegue de múltiples contratos inteligentes, uno para cada votación que se desee crear. Utilizando el bytecode y el ABI del contrato inteligente, se pueden desplegar nuevos contratos para cada nueva votación.

4.2 Oráculo

En el sistema de votación descentralizado, el oráculo juega un papel crucial en la integridad y funcionalidad del proceso. En este contexto, el oráculo se utiliza para subir los datos de los votos a la red IPFS, asegurando que cada voto emitido se registre de manera segura y los usuarios puedan comprobar su voto emitido.

El oráculo actúa como un puente entre el mundo off-chain y on-chain, permitiendo que los contratos inteligentes, que operan en la blockchain, interactúen con datos externos. Esto es esencial para registrar los votos en IPFS, ya que la blockchain por sí sola no puede acceder ni interactuar con redes externas directamente.

Utilizando un oráculo descentralizado, el sistema garantiza que los datos de los votos sean obtenidos y procesados de manera segura y sin manipulación. El oráculo se asegura de que el archivo que contiene los datos del voto se suba a IPFS de forma segura y que esté cifrado antes de su publicación. Esto añade una capa adicional de seguridad y verificabilidad al sistema, asegurando que solo los usuarios autorizados puedan acceder a la información sensible.

El diseño de este oráculo es fruto de un trabajo de investigación que ha culminado en un artículo aceptado en el congreso BCCA 2024 (The Sixth International Conference on Blockchain Computing and Applications) [Gar+24], uno de los eventos más importantes a nivel mundial sobre blockchain, que este año tendrá lugar en Dubái, 26-29 Noviembre 2024. El oráculo presentado en el artículo se utilizó para generar números aleatorios seguros utilizando el algoritmo Fortuna [FS03], un Generador Criptográfico de Números Pseudoaleatorios (CPRNG) diseñado para proporcionar una fuente de números pseudoaleatorios de alta calidad, en cada uno de los nodos, lo que implica diferencias notables respecto a su uso final en el TFM.

El oráculo propuesto funciona siguiendo el patrón de solicitud-respuesta propuesto en los documentos de referencia de Ethereum [24e] y consta de dos partes:

- Contrato del oráculo: Es un contrato inteligente escrito en Solidity que escucha las solicitudes de datos de otros contratos, retransmite las consultas de datos al nodo del oráculo y difunde los datos devueltos a los contratos clientes. El contrato del oráculo expone algunas funciones que los contratos clientes invocan al hacer una solicitud de datos.
- Nodo del oráculo: Es el componente fuera de la cadena del servicio de oráculo. Este nodo realizará las operaciones off-chain y transmitirá al contrato del oráculo los resultados para que puedan ser utilizados por los contratos clientes.

Este diseño modular del sistema permitirá actualizar y mejorar cada componente de manera independiente, sin afectar la funcionalidad general del sistema.

La comunicación entre el contrato del oráculo y el nodo del oráculo se realizará a través de eventos. Cuando el contrato del oráculo reciba una llamada, emitirá un evento. Al ocurrir un evento dentro del contrato inteligente, se generará un

registro de ese evento en la blockchain. Este registro incluirá los datos relevantes obtenidos de la llamada al contrato del oráculo y se almacenará permanentemente en la blockchain, permitiendo que el nodo del oráculo lo consulte.

Para que el servicio del oráculo funcione adecuadamente, el nodo del oráculo debe monitorizar constantemente la red de Ethereum en la cual se ha desplegado el contrato del oráculo. Para ello, será necesario suscribirse al evento utilizando la dirección del contrato y su ABI. Cuando el contrato del oráculo emita un evento, el nodo del oráculo lo detectará, realizará una operación basada en los argumentos del evento y lo enviará al contrato del oráculo mediante una transacción. A través del evento, se pueden proporcionar diferentes valores al nodo del oráculo.

El funcionamiento del oráculo se describe en el diagrama de secuencia mostrado en la Fig. 4.1. A continuación se detallan los pasos principales de operación:

1. Primero, un contrato externo que desea utilizar el oráculo llama a la función del contrato del oráculo.
2. Esta función emitirá un evento con los datos necesarios, que se registrará en la blockchain.
3. El nodo del oráculo, que monitoriza la blockchain en espera de eventos, detectará el evento y obtendrá los datos necesarios.
4. El nodo del oráculo realizará una acción según los requisitos especificados en los datos del evento.
5. El nodo del oráculo enviará una transacción según los requisitos especificados en los datos del evento.
6. El contrato externo accederá nuevamente a las variables del contrato del oráculo para verificar si los valores han cambiado.

Al elegir el lenguaje para implementar el contrato del oráculo, se seleccionó Solidity, ya que es el lenguaje de programación más popular para la Máquina Virtual de Ethereum (EVM). En el List. 4.6 se muestra el código del smart contract.

Dentro del contrato inteligente del oráculo, encontramos el evento 'IPFS_Add_Event' y la función 'trigger_IPFS_Add_Event'. Esta función será llamada desde nuestro sistema, proporcionándole los datos que queremos almacenar en la red IPFS. Al ejecutar esta función, se emitirá el evento con los datos correspondientes a la red, permitiendo que los nodos del oráculo puedan leer la información y subir el archivo correspondiente a la red IPFS.

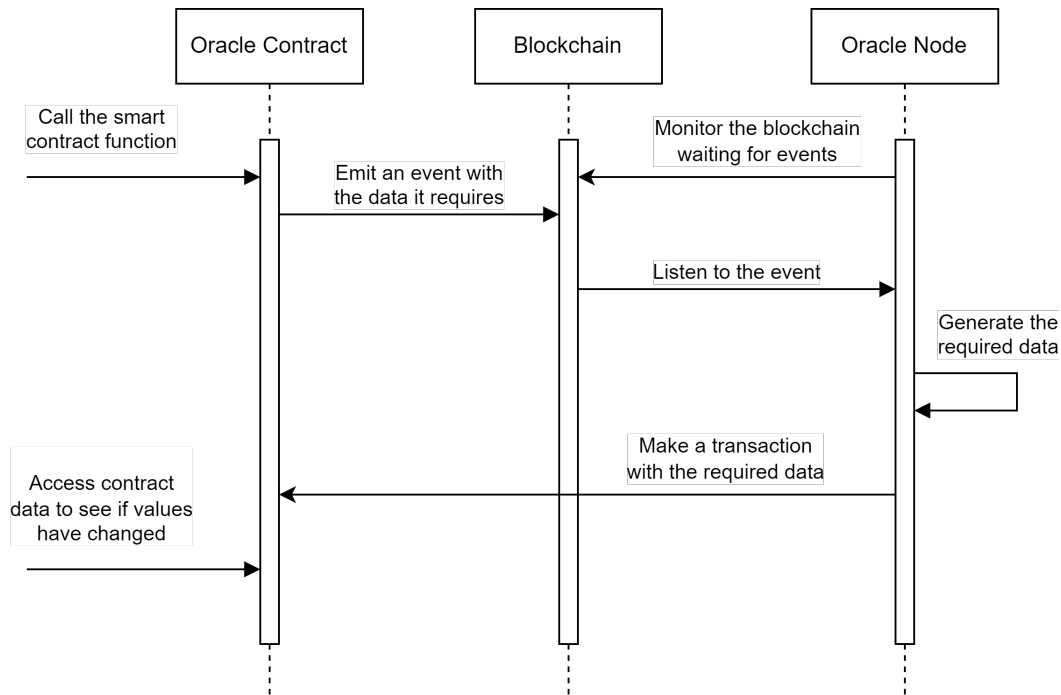


Figura 4.1.: Diagrama de secuencia del funcionamiento del oráculo (fuente: [Gar+24]).

Por otra parte, cuando deseemos recuperar el archivo de la red IPFS, utilizaremos la función ‘trigger_IPFS_Cat_Event’, que llamará al evento ‘IPFS_Cat_Event’ y recuperará el archivo de la red IPFS.

El cifrado y descifrado de los datos del archivo que se subirá a IPFS se realizará antes de llamar al oráculo en el caso de la subida del archivo, y después de recuperarlo en el caso de la descarga. De esta manera, aseguramos que todos los datos almacenados en la red IPFS estén cifrados. Para el cifrado, se ha tomado la decisión de utilizar AES-128, ya que lo consideramos perfecto por el equilibrio entre seguridad y rendimiento que aporta para este escenario.

List. 4.6: Smart contract del oráculo.

```

1 // SPDX-License-Identifier: MIT
  pragma solidity ^0.8.0;

  contract Oraculo {
5
    event IPFS_Add_Event(address indexed _sender, string cipher);
    function trigger_IPFS_Add_Event( string memory _cipher) external {
        // Emitir el evento
        emit IPFS_Add_Event(msg.sender, _cipher);
10 }

    event IPFS_Cat_Event(address indexed _sender);
    function trigger_IPFS_Cat_Event() external {
  
```



```

    emit IPFS_Cat_Event(msg.sender);
15 }

    mapping(string => string) private userToCipher;
    function setValuesIPFS(string memory _userAddress, string memory
                                                _cipher) public {
        userToCipher[_userAddress] = _cipher;
20 }

    function getValuesIPFS(string memory _userAddress) public view
                                                returns (string memory)
    {
        return userToCipher[_userAddress];
    }
25 }

```

Durante la implementación del nodo del oráculo, se decidió desarrollar un programa en Python utilizando la biblioteca web3 para suscribirse a los eventos del contrato del oráculo. Este enfoque requiere establecer una conexión con la red de Ethereum donde se desplegó el contrato, junto con su dirección y ABI asociadas. Al adoptar esta metodología, el nodo puede monitorizar y responder a los eventos del contrato en tiempo real de manera efectiva.

En el List. 4.7 se muestra una sección del código del nodo del oráculo, donde las primeras líneas ilustran cómo se establece la conexión con la red de Ethereum, así como la conexión a la dirección del contrato inteligente. Para llevar a cabo esta conexión, es necesario conocer tanto la dirección en la que el contrato está desplegado como el ABI del mismo. El ABI actúa como una interfaz que permite que el nodo del oráculo interactúe con las funciones y eventos del contrato inteligente.

List. 4.7: Código de la conexión con la red Ethereum.

```

1 # Se importa la librería que permite establecer la conexión con la red
    Ethereum
    from web3 import Web3

    # Conexión a la red de Ethereum usando Hardhat (o cualquier otro
        proveedor de nodo)
5 web3= Web3(Web3.HTTPProvider('http://127.0.0.1:8545/'))

    # Dirección del smart contract desplegado
    contract_address= Web3.to_checksum_address("...")

10 # ABI del smart contract
    contract_abi= json.loads('...')

    # Crear una instancia del smart contract
    contract= web3.eth.contract(address=contract_address, abi=contract_abi)

```

En este fragmento de código, se establece una conexión con un nodo local de Ethereum a través de Web3.HTTPProvider, especificando la URL del nodo. Luego, se crea una instancia del contrato inteligente usando la dirección en la que está desplegado y el ABI del contrato. Esta instancia es fundamental para invocar las funciones del contrato y escuchar los eventos emitidos.

La función `handle_event_add_IPFS` se muestra en el List. 4.8 y es responsable de gestionar el evento `IPFS_Add_Event`, que se activa cuando el contrato inteligente emite este evento para indicar que un nuevo archivo debe ser subido a la red IPFS.

List. 4.8: Código de la subida del fichero.

```
1 def handle_event_add_IPFS(event):
    sender = event['args']['_sender']
    cipher = event['args']['cipher']

5     print(f"Evento IPFS Add detectado - Remitente: {sender}")

    # Se lleva a cabo la transacción
    try:
        tx_hash = contract.functions.setValuesIPFS( cipher).transact()
10        print(f"Transacción enviada: {Web3.to_hex(tx_hash)}")
        print(f"Usuario: {sender}")
        print(f"Fichero encriptado: {cipher}")
    except Exception as e:
        print(f"Error al enviar transacción: {e}")

15
    data = {}
    data['sender'] = sender
    data['cipher'] = cipher

20    with open('data.json', 'w', encoding='utf-8') as f:
        json.dump(data, f, ensure_ascii=False, indent=4)

    # Subir el fichero a la red IPFS
    api = Connect()
25    cid = IPFS.add(api, 'data.json')

    # Almacenar el CID en un diccionario con la dirección de la cuenta
    dict[sender] = cid

30    # Ejecutar una consulta a IPFS para comprobar que el objeto se ha
        subido correctamente

    path = "/ipfs/" + cid
    text = IPFS.cat(api, path)
```

En esta función, se maneja el evento `IPFS_Add_Event` para realizar varias acciones cruciales. Primero, se extraen los datos del evento, que incluyen el contenido del archivo encriptado (`cipher`) y la dirección de la cuenta (`sender`). A continuación, se conecta a un cliente IPFS local para agregar el archivo a la red IPFS. La respuesta

del cliente IPFS incluye un CID (Content Identifier) único que representa el archivo subido. Se realiza una verificación para asegurar que el contenido del archivo recuperado desde IPFS coincide con el contenido original. Finalmente, el CID del archivo se almacena en un diccionario (dict), asociándolo con la dirección de la cuenta que originó la solicitud.

La función `handle_event_cat_IPFS` se muestra en el List. 4.9 y es responsable de gestionar el evento `IPFS_Cat_Event`, el cual es emitido por el contrato inteligente para indicar que un archivo debe ser recuperado.

List. 4.9: Código de la descarga del fichero.

```
1 def handle_event_cat_IPFS(event):
    sender = event['args']['_sender']

    print(f"Evento IPFS Cat detectado - Remitente: {sender}")

5     # Se obtiene el cid del archivo desde el diccionario
    cid = dict[sender]

    # Establecer la conexión a IPFS
10    api = Connect()

    # Ejecutar una consulta a IPFS para obtener el archivo
    path = "/ipfs/" + cid
    object = IPFS.cat(api, path)
```

En esta función, primero se extrae la dirección de la cuenta (`sender`) desde los datos del evento. Luego, se obtiene el CID del archivo (`cid`) desde el diccionario (`dict`), se establece una conexión con un cliente IPFS local, y se recupera el contenido del archivo asociado con el CID especificado.

La función `watch_events` se muestra en el List. 4.10 y es fundamental para que el nodo del oráculo permanezca monitorizando constantemente la red Ethereum, detectando eventos emitidos por el contrato del oráculo y ejecutando las acciones correspondientes en función de esos eventos.

List. 4.10: Código de la función para monitorizar la red.

```
1 def watch_events():
    # Filtrar eventos
    event_filter_IPFS = contract.events.IPFS_Add_Event.create_filter(
        fromBlock="latest")

5    event_filter_IPFS_cat = contract.events.IPFS_Cat_Event.
        create_filter(fromBlock="latest")

    while True:
        for event in event_filter_IPFS.get_new_entries():
```

```

        handle_event_add_IPFS(event)
10     for event in event_filter_IPFS_cat.get_new_entries():
        handle_event_cat_IPFS(event)

```

La función `watch_events` configura dos filtros de eventos para el contrato inteligente, uno para el evento `IPFS_Add_Event` y otro para el evento `IPFS_Cat_Event`. Estos filtros permiten que el nodo del oráculo esté atento a los eventos emitidos por el contrato inteligente. Dentro de un bucle infinito, `watch_events` obtiene las nuevas entradas para cada filtro de evento y llama a las funciones correspondientes (`handle_event_add_IPFS` y `handle_event_cat_IPFS`) para procesar los eventos.

Cada uno de los nodos del oráculo deberá configurar y ejecutar un nodo de la red IPFS, que en este caso será una red IPFS local en lugar de utilizar una red IPFS pública. Esta configuración local permitirá un mayor control y seguridad sobre los datos manejados dentro del sistema. En una red IPFS local, los nodos están restringidos a una red privada, esto significa que solo los nodos autorizados dentro de esta red privada pueden acceder y compartir archivos. De esta forma podemos tener mejor control de acceso y gestión de la red.

Para que el nodo del oráculo pueda realizar todas estas funciones y ser monitorizado fácilmente, se ha decidido implementarlo dentro de un contenedor Docker. De esta manera, su despliegue es mucho más sencillo y la detección de posibles errores en el nodo se facilita considerablemente. El script mostrado en el List. 4.11 despliega el nodo de IPFS a la vez que ejecuta el nodo del oráculo en el contenedor.

List. 4.11: Código del script para el despliegue del nodo.

```

1  #!/bin/sh

    # Inicializar el nodo
    ipfs init

5     # Borrar el bootstrap por defecto
    ipfs bootstrap rm --all

    # Generar swarm.key
10 git clone https://github.com/Kubuxu/go-ipfs-swarm-key-gen
    cd go-ipfs-swarm-key-gen/
    cd ipfs-swarm-key-gen/
    go run main.go > /root/.ipfs/swarm.key

15 cd /app/
    python3 oracleNode.py & ipfs daemon

```

En la figura 4.2 se puede apreciar el diagrama de secuencia que ilustra el funcionamiento final del oráculo. A continuación se explican en detalle los procesos de subida y recuperación de un archivo desde el front-end.

Proceso de subida de un archivo:

1. El usuario inicia una solicitud de subida de archivo desde la interfaz del front-end.
2. El front-end prepara los datos del archivo y los cifra para asegurar su confidencialidad.
3. El front-end envía una transacción al contrato inteligente, invocando la función `trigger_IPFS_Add_Event` y proporcionando los datos cifrados del archivo.
4. Esta función emite el evento `IPFS_Add_Event` con los datos necesarios para que los nodos del oráculo puedan procesar la solicitud.
5. El nodo del oráculo, que está monitorizando continuamente la red Ethereum mediante la función `watch_events`, detecta el evento `IPFS_Add_Event`.
6. Se ejecuta la función `handle_event_add_IPFS`, extrayendo los datos del evento.
7. El nodo del oráculo se conecta al cliente IPFS y sube el archivo cifrado a la red IPFS.
8. Se verifica que el archivo se ha subido correctamente y se obtiene el CID (Content Identifier) del archivo.
9. El nodo del oráculo almacena el CID del archivo en un diccionario, asociándolo con la dirección de la cuenta que realizó la llamada al contrato.

Proceso de recuperación de un archivo:

1. El usuario inicia una solicitud de recuperación de archivo desde la interfaz del front-end.
2. El front-end envía una transacción al contrato inteligente, invocando la función `trigger_IPFS_Cat_Event`.
3. Esta función emite el evento `IPFS_Cat_Event` con la dirección de la cuenta que solicita la recuperación.
4. El nodo del oráculo, que está monitorizando continuamente la red Ethereum mediante la función `watch_events`, detecta el evento `IPFS_Cat_Event`.
5. Se ejecuta la función `handle_event_cat_IPFS`, extrayendo los datos del evento.

6. El nodo del oráculo busca el CID del archivo en un diccionario, asociado con la dirección de la cuenta que realizó la llamada al contrato.
7. El nodo del oráculo se conecta al cliente IPFS y recupera el archivo cifrado de la red IPFS.
8. El nodo del oráculo envía los datos del archivo cifrado al contrato del oráculo.
9. El front-end accede a los datos del contrato del oráculo y recupera los datos.
10. Se descifran los datos y se recupera el fichero original.

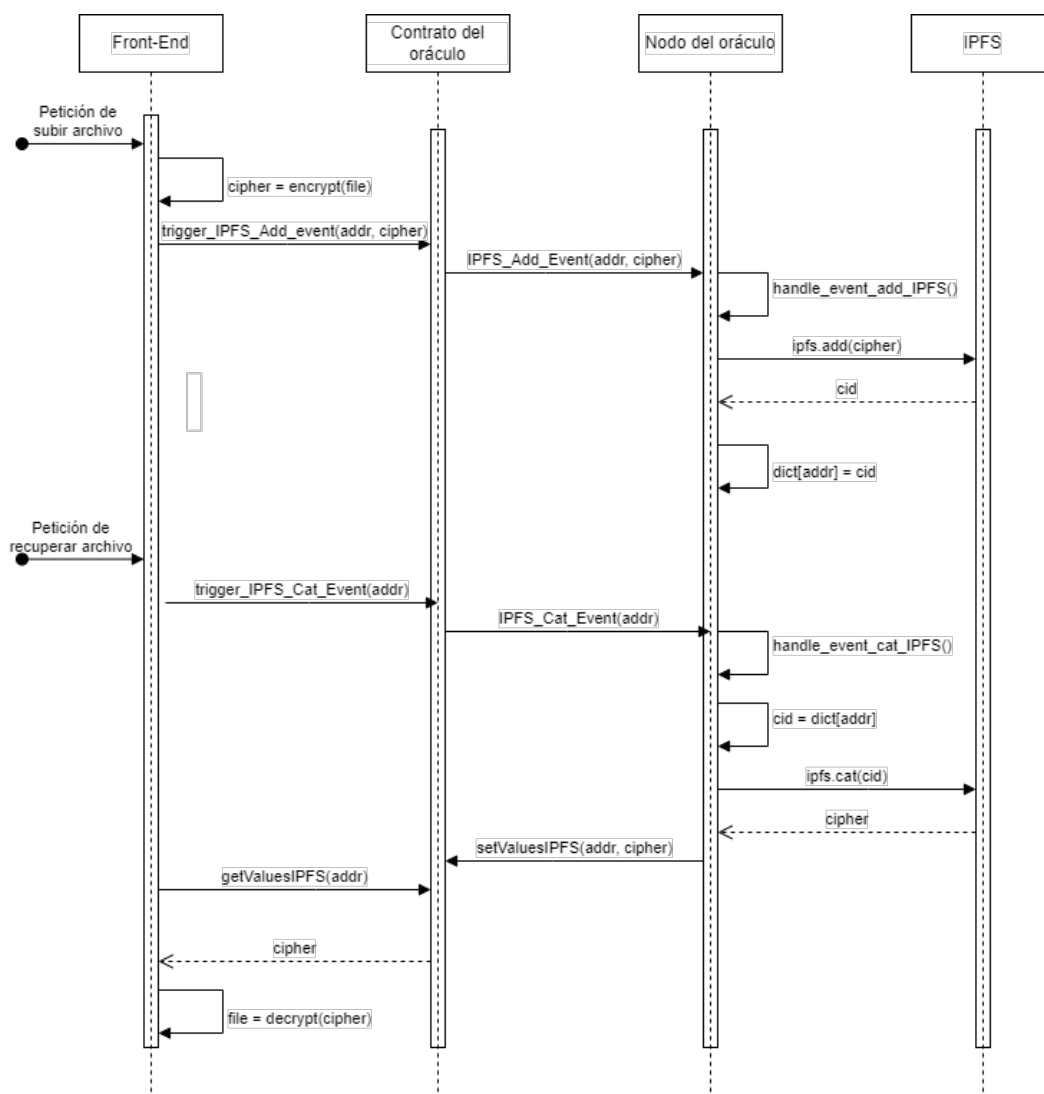


Figura 4.2.: Diagrama de secuencia del funcionamiento del oráculo.

4.3 OrbitDB

En el caso de OrbitDB, la implementación ha sido complicada debido a la escasa documentación disponible. OrbitDB es una tecnología nueva y en constante evolución, lo que provoca que la documentación a menudo esté desactualizada y que las funciones cambien entre versiones. Esto ha dificultado el proceso de implementación, sin embargo, con el tiempo y la maduración de OrbitDB, se espera que la documentación mejore y se estabilice.

En nuestro sistema utilizaremos OrbitDB para almacenar las direcciones de todos los contratos inteligentes de votaciones abiertas simultáneamente. Esta lista de direcciones será recuperada por el front-end mediante un API (Application Programming Interface) REST (Representational State Transfer). Para almacenar la lista de direcciones, se utilizará una base de datos de OrbitDB de tipo clave-valor, almacenando la dirección del contrato como clave y el nombre de la votación como valor, de forma que se pueda realizar una búsqueda del contrato específico si se conoce el nombre de la votación.

Se publicará en IPFS una lista de direcciones de las APIs de cada nodo disponible. De esta forma, se puede utilizar una API de manera aleatoria y, en caso de que el nodo seleccionado aleatoriamente no esté disponible en ese momento, simplemente se repetirá el proceso y se realizará la petición a otro nodo aleatorio.

La utilización de OrbitDB como base de datos para almacenar las direcciones de los contratos inteligentes desplegados en el sistema de votación es de suma importancia por varias razones. OrbitDB ofrece ventajas significativas en comparación con las alternativas tradicionales de almacenamiento centralizado y otras soluciones descentralizadas, proporcionando un entorno más seguro, robusto y eficiente.

Una de las principales ventajas de OrbitDB es su naturaleza descentralizada. A diferencia de las bases de datos centralizadas, donde un único servidor actúa como punto de control y puede ser un punto de fallo, OrbitDB replica los datos a través de múltiples nodos. Esto significa que incluso si algunos nodos fallan o se desconectan, los datos siguen siendo accesibles desde otros nodos de la red, garantizando una alta disponibilidad y reduciendo el riesgo de pérdida de datos.

En comparación con las bases de datos centralizadas como MySQL o PostgreSQL, OrbitDB ofrece una solución más segura y resistente para aplicaciones críticas como un sistema de votación. Las bases de datos centralizadas pueden ser vulnerables a ataques de denegación de servicio (DoS) y a fallos de hardware, mientras que la

naturaleza distribuida de OrbitDB mitiga estos riesgos. OrbitDB proporciona un conjunto robusto de características que hacen que sea la mejor opción para almacenar las direcciones de los contratos inteligentes en un sistema de votación descentralizado.

Además, dejando de lado los posibles ataques, debemos considerar cómo funcionan normalmente los sistemas de votación, los cuales tienen que soportar una alta capacidad de conexiones simultáneas. Generalmente, estos procesos están abiertos durante un día o, en algunos países, durante un fin de semana. OrbitDB nos proporciona la capacidad de escalar el sistema en función del número de votantes esperados, mejorando así su disponibilidad. A medida que se añaden más nodos a la red, la capacidad de almacenamiento y procesamiento aumenta de manera eficiente sin depender de un único servidor central.

A continuación, se detalla parte del código de los nodos de OrbitDB, explicando paso a paso cada una de las funciones.

Primero, se crea una instancia de LevelBlockstore, que es el componente encargado de almacenar los bloques de datos de IPFS en la carpeta `./ipfs/blocks`. Luego, se configura y se inicializa libp2p, una biblioteca para la red descentralizada que IPFS utiliza para permitir la comunicación entre los nodos de la red. Después de configurar libp2p, se crea una instancia de IPFS usando la biblioteca Helia [24g], una biblioteca para interactuar con la red IPFS. Esta instancia de IPFS se establece con libp2p y blockstore. A continuación, se crea una instancia de OrbitDB a partir de Helia, y sobre esta instancia se crea una base de datos key-value denominada my-db, que almacenará datos en forma de pares clave-valor. En el almacenamiento clave-valor, se utilizará como clave la dirección del contrato desplegado en la blockchain y como valor un título, el cual será una cadena de texto proporcionada por el administrador del contrato. Este título servirá para identificar la votación de manera clara y sencilla para los usuarios. Finalmente, se imprime en la consola la dirección de la base de datos my-db. Esta dirección es única y se utiliza para que otros nodos puedan conectarse a la misma base de datos. Esto se muestra en el List. 4.12.

List. 4.12: Conexión del nodo OrbitDB con la red IPFS.

```
1 const blockstore = new LevelBlockstore('./ipfs/blocks');  
const libp2p = await createLibp2p(Libp2pOptions);  
const ipfs = await createHelia({ libp2p, blockstore });  
5 const orbitdb = await createOrbitDB({ ipfs });
```


En el primer nodo de la red, se creará una instancia de OrbitDB a partir de Helia, y sobre esta instancia se crea una base de datos key-value denominada my-db, que almacenará datos en forma de pares clave-valor. En el almacenamiento clave-valor, se utilizará como clave la dirección del contrato desplegado en la blockchain y como valor un título, el cual será una cadena de texto proporcionada por el administrador del contrato. Este título servirá para identificar la votación de manera clara y sencilla para los usuarios. Finalmente, se imprime en la consola la dirección de la base de datos my-db. Esta dirección es única y se utiliza para que otros nodos puedan conectarse a la misma base de datos. Esto se muestra en el List. 4.13.

List. 4.13: Creación de la base de datos OrbitDB.

```
1 const db = await orbitdb.open('my-db', { type: 'keyvalue' });  
  
  console.log('my-db address', db.address);
```

En el resto de nodos que se conecten a la red, en lugar de las líneas anteriores, se sustituirán por las mostradas en el List. 4.14.

List. 4.14: Conexión del nodo OrbitDB con la base de datos OrbitDB.

```
1 // Conexión al nodo principal  
  const mainNodeMultiaddr = multiaddr('/ip4/127.0.0.1/tcp/34422/p2p/  
                                     12D3Koo...');  
  
  await libp2p.dial(mainNodeMultiaddr);  
  
5 // Abrir la base de datos  
  const dbAddress = '/orbitdb/zdpu...';  
  const db = await orbitdb.open(dbAddress, { type: 'keyvalue' });
```

Aquí se utiliza una dirección multiaddr para conectarse al nodo principal. Esta dirección incluye detalles como la dirección IP (127.0.0.1), el puerto TCP (34422), y el identificador del nodo (12D3Koo...). De esta manera, se inicia una conexión al nodo principal usando la dirección multiaddr. Una vez establecida la conexión con el nodo principal, se procede a abrir la base de datos utilizando la dirección que se mostró en la consola del primer nodo. Esta dirección es una cadena única que identifica la base de datos en la red IPFS y OrbitDB. La base de datos se abre para poder realizar operaciones de lectura y escritura sobre ella, permitiendo así interactuar con los datos almacenados en el sistema.

Para facilitar la comunicación entre el navegador y los nodos, se utiliza un API. Este API actúa como un puente entre el cliente (navegador) y los nodos de la red blockchain, permitiendo que las solicitudes y respuestas se gestionen de manera eficiente y estructurada.

El API permite a los clientes enviar solicitudes HTTP GET a la ruta /records para obtener todos los registros almacenados en una base de datos OrbitDB. La respuesta se envía en formato JSON. Esto se muestra en el List. 4.15.

List. 4.15: Endpoint para obtener todos los registros.

```
1 app.get('/records', async (req, res) => {  
    const records = await db.all();  
    res.json(records);  
});
```

Enviando solicitudes HTTP POST a la ruta /records, los usuarios pueden añadir un nuevo registro a la base de datos. Los usuarios envían una clave y un valor, y el endpoint almacena este par en la base de datos. Se devuelve al usuario el par clave-valor almacenado y el hash que lo identifica dentro de la base de datos. Esto se muestra en el List. 4.16.

List. 4.16: Endpoint para añadir un nuevo registro.

```
1 app.post('/records', async (req, res) => {  
    const { key, value } = req.body;  
    const hash = await db.put(key, value);  
  
5    res.json({ key, value, hash });  
});
```

El endpoint /records/search permite a los usuarios filtrar los registros por un valor específico. Los usuarios pueden buscar entre todos los registros para encontrar aquellos que tienen un valor que coincide con el valor proporcionado en la consulta. Como valor para buscar, se utiliza el título almacenado. Esto se muestra en el List. 4.17.

List. 4.17: Endpoint para filtrar registros por valor.

```
1 app.get('/records/search', async (req, res) => {  
    const searchValue = req.query.value;  
    const records = await db.all();  
  
5    const filteredRecords = records.filter(record => record.value ===  
                                                searchValue);  
    res.json(filteredRecords);  
});
```

4.4 Front-End

El front-end del sistema es la parte principal que integra el resto de los componentes. Para ello, se ha decidido implementar una aplicación web basada en Node.js.

Node.js es la mejor opción para programar el front-end de una DApp por varias razones:

- Escalabilidad: Node.js tiene capacidad de manejar un gran número de conexiones simultáneas de manera eficiente, lo cual es crucial para una DApp que debe soportar múltiples usuarios interactuando con la blockchain en tiempo real.
- Ecosistema de paquetes: La extensa colección de paquetes y módulos disponibles a través de npm (Node Package Manager) facilita la integración de diversas funcionalidades necesarias para una DApp. Entre estos, la biblioteca web3.js es fundamental para interactuar con la blockchain, manejar la autenticación de usuarios, interactuar con contratos inteligentes, y gestionar el estado de la aplicación.
- Facilidad de uso con MetaMask: Node.js facilita la integración con MetaMask, una herramienta esencial para interactuar con la blockchain desde el navegador. Esta compatibilidad simplifica la autenticación y las transacciones de los usuarios, mejorando la experiencia general.

Además, Node.js también ofrece varias características y prácticas recomendadas que ayudan a asegurar la aplicación:

- Node.js y sus paquetes son actualizados periódicamente para proteger la aplicación contra vulnerabilidades conocidas.
- Herramientas como npm audit permiten identificar y solucionar vulnerabilidades en las dependencias de la aplicación.
- Node.js permite utilizar tokens CSRF (Cross-Site Request Forgery) para proteger la aplicación contra este tipo de ataques.
- Node.js permite asegurar que todas las comunicaciones entre el cliente y el servidor estén cifradas utilizando HTTPS, protegiendo así los datos durante la transmisión.

Al utilizar la aplicación web por primera vez, el navegador solicitará que conectemos nuestra cartera de MetaMask. Esto permitirá realizar operaciones dentro de la web de manera segura y eficiente. A través de la aplicación, se realizarán llamadas a MetaMask para gestionar estas operaciones.

Una vez conectada la cartera de MetaMask, cada vez que se necesite realizar una transacción, la aplicación web enviará una solicitud a MetaMask, la cual mostrará

un desplegable con los detalles de la transacción. Este desplegable permitirá al usuario revisar y analizar los datos de la transacción antes de proceder. MetaMask proporciona una interfaz segura para que el usuario pueda firmar las transacciones, asegurando que todas las operaciones se realizan de manera segura y protegida.

Esta integración con MetaMask no solo facilita la autenticación y autorización de transacciones, sino que también garantiza la integridad y seguridad de las mismas. Al firmar las transacciones dentro de MetaMask, los usuarios pueden estar seguros de que sus claves privadas no son expuestas y que cada operación es validada de forma segura antes de ser enviada a la blockchain. Además, MetaMask también ofrece protección contra sitios maliciosos y phishing [24c], añadiendo una capa adicional de seguridad para los usuarios.

En el List. 4.18 se muestra la función `loadWeb3`, que comprueba si MetaMask está instalado y es compatible (`window.ethereum`). Si es compatible, solicita al usuario que conceda acceso a sus cuentas de MetaMask. Si no está disponible `window.ethereum`, intenta usar `window.web3` para compatibilidad con versiones anteriores. Si ninguno de los dos está disponible, alerta al usuario que necesita instalar MetaMask. Esta función es crucial para permitir interactuar con la blockchain Ethereum a través del navegador y MetaMask.

List. 4.18: Función `loadWeb3`.

```
1 async function loadWeb3() {  
    if (window.ethereum) {  
        web3 = new Web3(window.ethereum);  
        await window.ethereum.request({ method: 'eth_requestAccounts' }  
                                     );  
5    } else if (window.web3) {  
        web3 = new Web3(window.web3.currentProvider);  
    } else {  
        alert('You need to install MetaMask!');  
    }  
10 }
```

Una vez que la cartera de MetaMask esté conectada a la aplicación web, los usuarios tendrán acceso a una variedad de funciones y casos de uso, que estarán determinados por su rol específico dentro del sistema.

Los administradores, que son responsables de gestionar el sistema de votación, tendrán varias capacidades una vez que su cartera esté conectada. En primer lugar, podrán desplegar los contratos inteligentes necesarios para las votaciones. Este proceso incluye la creación y configuración de nuevos contratos, así como la definición de los parámetros de cada votación, como las fechas de inicio y fin, los candidatos o propuestas, y otros detalles esenciales para la gestión del proceso electoral.

En el List. 4.19 se muestra la función `deployContract`, que permite a los administradores desplegar los smart contracts de las votaciones. La función `deployContract` obtiene la cuenta del usuario conectada a MetaMask, recolecta los datos de entrada del usuario desde el DOM y los valida (los parámetros que serán utilizados para crear el smart contract, como el título que identificará a la votación y las opciones de selección) y crea una nueva instancia del contrato utilizando el bytecode y el ABI. Si tiene éxito, almacena la dirección del contrato desplegado en la base de datos de OrbitDB y actualiza la interfaz de usuario en consecuencia.

List. 4.19: Función `deployContract`.

```
1 async function deployContract() {
    const accounts = await web3.eth.getAccounts();
    const account = accounts[0];

    5    const title = document.getElementById('titleInput').value;
    const option1 = document.getElementById('option1Input').value;
    const option2 = document.getElementById('option2Input').value;

    if (!title || !option1 || !option2) {
    10        document.getElementById('deployResult').innerText = 'Please
                                                    enter a title and both
                                                    options';

        return;
    }

    const VotingContract = new web3.eth.Contract(contractABI);

    15    try {
        const newContractInstance = await VotingContract.deploy({
            data: contractBytecode,
            arguments: [title, option1, option2]
        20        }).send({
            from: account,
            gas: 1500000,
            gasPrice: '30000000000'
        });

    25        document.getElementById('deployResult').innerText = Contract
                                                    deployed;

    28        // Store deployed contract address
    30        addRecord(newContractInstance.options.address, title);

        // Actualiza la lista de contratos activos
        updateDeployedContractsDropdown();
    } catch (error) {
    35        document.getElementById('deployResult').innerText = Error
                                                    deploying contract;

    36    }
}
```

Por otro lado, los votantes tendrán acceso a funcionalidades diseñadas para permitirles participar en el proceso electoral de manera eficiente. Una vez conectada su cartera de MetaMask, podrán votar en las elecciones que estén abiertas en ese momento. La aplicación web les mostrará una lista de votaciones activas y disponibles, permitiéndoles revisar las opciones, emitir su voto y confirmar su elección.

La función `voteForOption` mostrada en el List. 4.20 obtiene la cuenta del usuario conectada a MetaMask, determina la opción de votación seleccionada por el usuario basada en el parámetro `option` pasado a la función y envía una transacción al smart contract para registrar el voto utilizando la cuenta del usuario. Si tiene éxito, muestra un mensaje de confirmación al usuario.

List. 4.20: Función `voteForOption`.

```
1 async function voteForOption(option) {  
    const accounts = await web3.eth.getAccounts();  
    const account = accounts[0];  
  
5    console.log("Votando en la votación: ", document.getElementById('contractTitle').value);  
  
    if (option == 1){  
        option = document.getElementById('contractOption1').value  
    } else if (option == 2){  
        option = document.getElementById('contractOption2').value  
10    }  
  
    console.log("Valor: ", option);  
  
    try {  
15        await contractInstance.methods.vote(  
            option  
        ).send({  
            from: account,  
            gas: 1500000,  
20            gasPrice: '30000000000'  
        });  
        console.log('Vote submitted for option', option);  
        alert('Vote submitted successfully!');  
  
25        try {  
            await uploadStringToIPFS(option);  
        } catch (uploadError) {  
            console.error('Error en uploadStringToIPFS:', uploadError);  
        }  
30    } catch (error) {  
        console.error('Error voting:', error);  
        alert('Error voting: ' + error.message);  
    }  
35 }
```

Además, tanto los administradores como los votantes podrán consultar el número de votaciones actuales en curso. Esta funcionalidad les permite monitorizar el estado de las votaciones, verificar cuántas están activas en un momento dado y consultar el número total de votos emitidos hasta el momento. En el List. 4.21 se muestra la función `viewResults`, que permite obtener el número de votos actuales directamente del contrato inteligente llamando al método `getVotes`.

List. 4.21: Función `viewResults`.

```
1 async function viewResults() {  
    try {  
        const result = await contractInstance.methods.getVotes().call()  
        ;  
        alert('Votes for Option 1: {result[0]}\nVotes for Option 2: {  
            result[1]}' );  
5    } catch (error) {  
        alert('Error fetching results: ' + error.message);  
    }  
}
```

Para modificar el código y evitar que los usuarios con cuentas de administradores vean las opciones de votar, o que los usuarios con cuentas de votante vean las opciones restringidas a los administradores (como desplegar los contratos de votaciones o cerrar las votaciones), se ha agregado una verificación de la dirección de la cuenta conectada antes de mostrar las opciones. Esto se ha hecho utilizando la librería de JavaScript Web3 y una lista hardcodeada en el código que contiene las cuentas que utilizamos para nuestro escenario, simulando que son cuentas de administradores.

En un escenario real, esta lista no se encontraría hardcodeada, principalmente para permitir su fácil modificación en caso de tener que agregar nuevas cuentas de administrador, así como por motivos de seguridad. En un escenario real, podríamos utilizar los propios nodos de OrbitDB en la lista de votaciones abiertas (es decir, las direcciones de los contratos que se encuentran abiertos en ese momento) para que almacenen también la lista de cuentas que están autorizadas como administradores en el sistema.

Política de contraseñas

Al utilizar una contraseña para proteger mediante un cifrado los datos almacenados en IPFS, es crucial definir una política de contraseñas que tenga en cuenta varios aspectos de seguridad. Las contraseñas serán generadas aleatoriamente, de esta forma evitaremos problemas asociados comúnmente a las contraseñas generadas

Tipo de caracteres	Ejemplo	Conjunto de caracteres
Minúscula	a-z	26
Mayúscula	A-Z	26
Números	0-9	10
Caracteres especiales (símbolos)	!, ?, &, @, *, %, \$, #	8

Cuadro 4.1.: Posibles caracteres clasificados por grupos.

por humanos: el hecho de reutilizar contraseñas, utilizar información personal para las contraseñas (como como nombres, fechas de nacimiento o palabras comunes) o utilizar contraseñas secuenciales.

Como parte de la política de contraseñas, se buscará definir:

- El conjunto de caracteres disponibles.
- El número de caracteres presentes en la contraseña.
- La función de hash utilizada sobre las contraseñas.

Si utilizamos como posibles caracteres los caracteres mostrados en la tabla 4.1, tenemos un total de 70 posibles caracteres.

Para calcular las posibles combinaciones, podemos usar la siguiente fórmula: $s = n^l$, siendo s el número total de posibles combinaciones, n el número de caracteres en el conjunto de caracteres disponibles y l el número de caracteres presentes en la contraseña.

Por lo tanto, utilizando el conjunto de caracteres definidos anteriormente, y con una longitud de 16 caracteres, el número de posibles combinaciones sería de 70^{16} .

Para generar una contraseña de forma segura en JavaScript, utilizamos la API de criptografía nativa del navegador, llamada `window.crypto`. Esta API proporciona métodos para generar números aleatorios seguros. La función utilizada para generar las contraseñas se muestra en el List. 4.22.

List. 4.22: Función `generateSecurePassword`.

```

1 function generateSecurePassword(length) {
    // Se definen los caracteres que pueden ser utilizados en la
    //                                contraseña

    const charset = 'ABCDEFGHJKLM...';
    const charsetLength = charset.length;

5
    // Se crea un array para almacenar valores aleatorios
    const randomValues = new Uint32Array(length);

```



```

10 // Se llena el array con valores aleatorios criptográficamente
    seguros
    window.crypto.getRandomValues(randomValues);

    // Se convierten los valores aleatorios a caracteres utilizables
    const passwordArray = [];
    for (let i = 0; i < length; i++) {
15         const randomIndex = randomValues[i] % charsetLength;
        passwordArray.push(charset[randomIndex]);
    }

    // Se unen los caracteres en una cadena y se devuelven como
    contraseña
20     return passwordArray.join('');
}

```

Asumiendo un ataque de fuerza bruta donde se prueban todas las combinaciones posibles de caracteres hasta encontrar la contraseña correcta, se puede utilizar la siguiente fórmula para estimar el tiempo necesario:

$$T = N^L / R$$

Donde T es el tiempo estimado en segundos, N es el número de caracteres en el conjunto de caracteres disponibles, L el número de caracteres presentes en la contraseña y R es la tasa de intentos por segundo que el atacante puede realizar. Es necesario tener en cuenta que la tasa de intentos por segundo (R) dependerá de la capacidad de procesamiento del sistema que realiza el ataque, que dependerá también del hash utilizado para cifrar las contraseñas.

El resultado obtenido es el tiempo completo de búsqueda, es decir, durante este tiempo la contraseña será encontrada con un 100% de probabilidad. La probabilidad de encontrar la contraseña durante la mitad de este tiempo es del 50%.

4.4.1 Comunicación con el oráculo

Al emitir un voto se llamará automáticamente a la función `uploadStringToIPFS`, mostrada en el List 4.23, que será la encargada de realizar la conexión con el oráculo y subir el voto cifrado a la red IPFS.

List. 4.23: Función `uploadStringToIPFS`.

```

1 async function uploadStringToIPFS(inputString) {
    // Generar una contraseña segura de 16 caracteres
    const password = generateSecurePassword(16);
5

```

```

if (typeof password !== 'undefined' && password !== null && password !==
    = '') {
  if (typeof inputString === 'string' && inputString !== '') {

    // Asignar valor a la variable clave
10    const clave = password;
    console.log("Subiendo voto a IPFS con la contraseña: " + clave)
        ;

    // Cifrar el mensaje con la contraseña
    const mensajeCifrado = CryptoJS.AES.encrypt(inputString, clave)
        .toString();

15    // Crear una instancia del contrato oráculo
    const oracleContract = new web3.eth.Contract(oracleABI,
        oracleAddress);

    // Llamar al contrato oráculo para desencadenar el evento
20    oracleContract.trigger_IPFS_Add_Event(mensajeCifrado).then(
        async (cipher) => {
            // Mostrar la contraseña generada al usuario
            alert("Recibo del voto subido a IPFS con la contraseña: " +
                password);

        }).catch(error => {
            console.error("Error al desencadenar el evento IPFS_Add: ",
                error);

25    });
  } else {
    console.error("El string de entrada está vacío o no es válido."
        );
  }
} else {
30    console.error("El campo password está vacío o no definido.");
}

```

Pruebas

En esta sección, se detallarán las pruebas realizadas para verificar el correcto funcionamiento de cada componente del sistema. A continuación se explicará cada prueba llevada a cabo para asegurar que todos los elementos operan de manera eficiente y conforme a lo esperado.

5.1 Smart contract

Para asegurar la robustez y seguridad del smart contract que se utiliza para las votaciones, se realizaron pruebas exhaustivas en varios aspectos:

- Se probaron individualmente todas las funciones del contrato para verificar que se comporten según lo esperado. En la Fig. 5.1 se muestran los logs resultantes de cada una de estas pruebas, realizadas desde Remix IDE [24m]. En el recuadro rojo se muestra el resultado de llamar al constructor del Smart Contract. En el recuadro azul se presenta el resultado de llamar a la función de votación con una cuenta autorizada a votar. En el recuadro verde se muestran los logs correspondientes a llamar a la función de votación con una cuenta que ya ha votado, por lo que se produce el error y no se permite el voto. En el siguiente recuadro rojo encontramos los logs que se producen en el caso de que una persona que no es el owner del contrato intenta cerrar la votación. En el recuadro blanco encontramos los logs que se reproducen cuando el owner del contrato cierra la votación, y en el último recuadro morado encontramos los logs cuando una persona intenta votar y la votación ya ha sido cerrada correctamente.
- Se verificó que todos los eventos sean emitidos correctamente y contengan la información adecuada.
- Se probó la interacción entre el contrato inteligente y el front-end para asegurar que los datos se transmitan correctamente.
- Se intentó simular ataques comunes a contratos inteligentes, como overflow. Sin embargo, en versiones de Solidity a partir de la 0.8.0, el compilador de

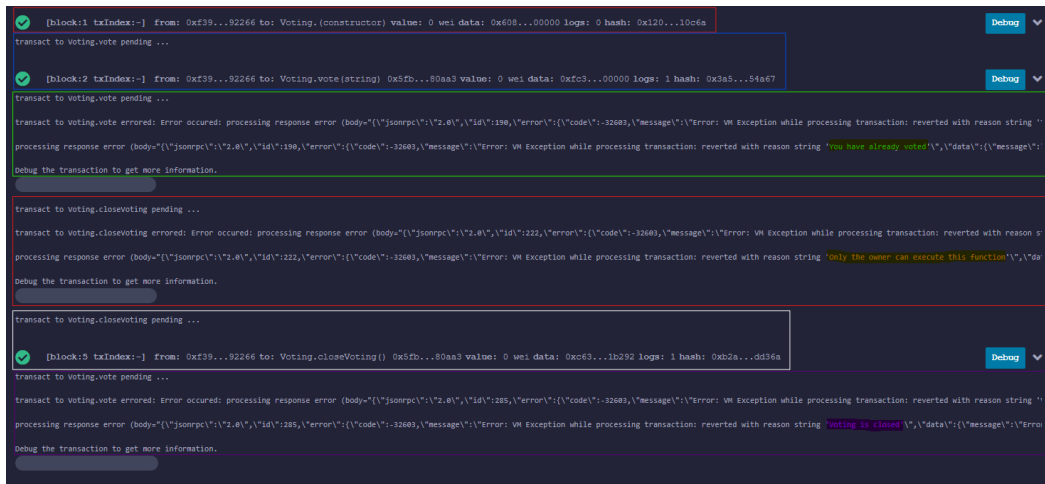


Figura 5.1.: Ejemplo de la ejecución de las pruebas del smart contract.

Solidity incluye comprobaciones automáticas de overflow y underflow, lanzando una excepción en caso de que se produzca, por lo que se comprobó que el contrato es resistente a esta clase de ataques.

5.2 Oráculo

El nodo del oráculo es un componente crítico, y su correcto funcionamiento es esencial para la integridad del sistema. Se realizaron las siguientes pruebas:

- Se verificó que el nodo del oráculo pueda conectarse de manera confiable a la red Ethereum y escuchar eventos.
- Se aseguró que el nodo del oráculo pueda interactuar con la red IPFS para subir y recuperar archivos.
- Se probó que el nodo del oráculo detecte y maneje correctamente los eventos emitidos por el contrato inteligente y se verificó que las funciones se ejecuten correctamente en respuesta a los eventos emitidos.
- Se realizaron pruebas para verificar que el nodo del oráculo pueda manejar múltiples solicitudes concurrentes sin degradar su desempeño.

En la Fig. 5.2 se puede apreciar parte de los logs generados en el contenedor que ejecuta el oráculo al recibir una petición para subir un fichero.

```

2024-07-18 11:25:03 Evento IPFS Add detectado - Remitente: 0xf39Fd6e51aad88F6F4ce6aB8827279cFfB92266
2024-07-18 11:25:03 QmdB11LR2fTggqt9K8gAVB9vsRDkiULyaLhz27hWsSTPwn
2024-07-18 11:25:03 {
2024-07-18 11:25:03     "sender": "0xf39Fd6e51aad88F6F4ce6aB8827279cFfB92266",
2024-07-18 11:25:03     "cipher": "hola"
2024-07-18 11:25:03 }

```

Figura 5.2.: Log del contenedor que ejecuta el nodo del oráculo al detectar un evento.

5.3 Nodos de OrbitDB

En el caso de los nodos de OrbitDB, se llevaron a cabo una serie de pruebas para asegurar su correcto funcionamiento y su capacidad para manejar diferentes escenarios:

- Se probó la funcionalidad del sistema añadiendo datos a OrbitDB a través de un nodo específico. Posteriormente, se intentó recuperar esos mismos datos a través de otro nodo diferente para verificar la consistencia y replicación de la base de datos. Esta prueba confirmó que los datos se replican correctamente entre nodos de OrbitDB.
- Se configuró el sistema con dos nodos de OrbitDB, uno activo y otro simulado como caído. Se probó la funcionalidad del sistema para asegurarse de que el sistema continúe operando correctamente a pesar de la caída de uno de los nodos.
- Se realizaron operaciones de lectura y escritura concurrentes en la base de datos desde diferentes nodos.
- Se implementaron y probaron mecanismos de autenticación para asegurar que solo los nodos autorizados puedan acceder y modificar los datos en OrbitDB.

En las Figs. 5.3 y 5.4 se puede ver cómo se desplegó un escenario de prueba compuesto por dos nodos OrbitDB conectados entre ellos. Cada uno de ellos despliega un front-end en un puerto diferente (el 3000 y el 3001). Ambos son capaces de obtener y almacenar los mismos datos gracias a la conexión desde la red IPFS y OrbitDB.

```
PS C:\Users\pedro\OneDrive\Escritorio\OrbitDB-Test\NodoPrincipal> node .\server.js
my-db address /orbitdb/zdpuAuxN6Druj6zXu9ANa9gcdeEVYQKwX6tFkUmgY7wSAE1wM

MainNodeId: 12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW
Nodo principal multiaddrs: /ip4/172.23.224.1/tcp/55434/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW
/ip4/192.168.46.1/tcp/55434/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW
/ip4/192.168.56.1/tcp/55434/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW
/ip4/192.168.138.1/tcp/55434/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW
/ip4/192.168.8.109/tcp/55434/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW
/ip4/127.0.0.1/tcp/55434/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW
Server running at http://localhost:3000

PS C:\Users\pedro\OneDrive\Escritorio\OrbitDB-Test\NodoSecundario> node .\client.js
Conectado al nodo principal en: /ip4/172.23.224.1/tcp/55434/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW/p2p/12D3KoolWJ3WhfHvqy5jQKwRxCxQWYsQsmcTjcyMEkdpPMedRiEW
Direccion de la base de datos: /orbitdb/zdpuAuxN6Druj6zXu9ANa9gcdeEVYQKwX6tFkUmgY7wSAE1wM
Server running at http://localhost:3001
```

Figura 5.3.: Logs de los nodos de la red de prueba de OrbitDB.

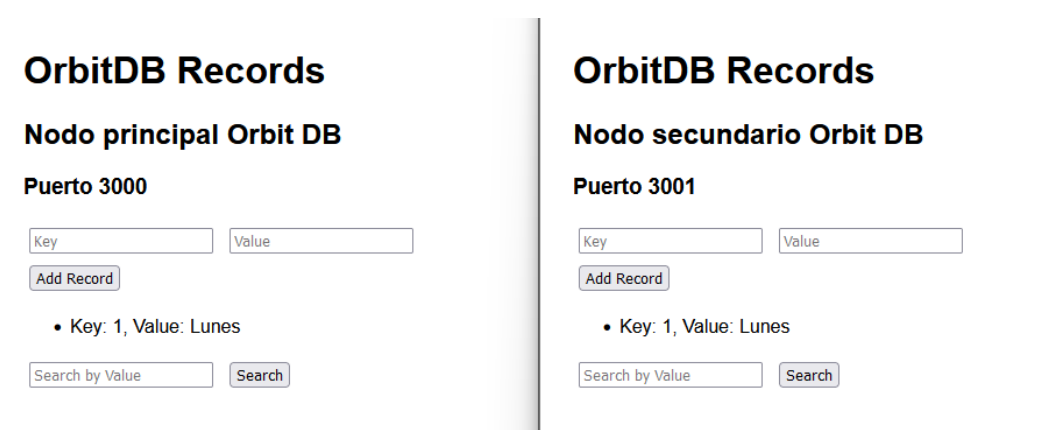


Figura 5.4.: Front-end de los nodos de la red de prueba de OrbitDB.

5.4 Front-End

Para asegurar el correcto funcionamiento del front-end, se llevaron a cabo una serie de pruebas en las que se aprobó el sistema global y su funcionamiento a través del navegador. Una vez ya comprobado el correcto funcionamiento de los componentes individuales, estas pruebas permitieron evaluar el sistema en su totalidad, asegurando que todo funcione de manera integrada y coherente a través del navegador. A continuación, se detallan las pruebas realizadas:

- Se evaluó la facilidad de navegación a través de la interfaz, asegurando que los usuarios puedan acceder a todas las funcionalidades de manera intuitiva.
- Se probó la integración de los smart contracts con MetaMask, asegurando que todas las funciones de los smart contracts eran accesibles y funcionales.

En el anexo B se detallan las instrucciones para desplegar el sistema, así como un walkthrough en el que se detalla el funcionamiento final del sistema y su utilización a través del front-end.

Resultados

6.1 Evaluación de la seguridad

En esta sección, se realiza una exhaustiva evaluación de la seguridad del sistema de votación tras su implementación. Se analizarán los mecanismos de seguridad proporcionados por las diversas tecnologías utilizadas para garantizar que los usuarios puedan votar de manera segura y confiable. Además, se detallarán las medidas de seguridad adicionales que, aunque no se implementaron debido a limitaciones de tiempo o capacidad de procesamiento, han sido diseñadas y planificadas para su futura incorporación en el sistema.

6.1.1 Aspectos a tener en cuenta

Al llevar a cabo este trabajo, se han tomado ciertas decisiones que, aunque no son las más adecuadas para la seguridad del sistema, se han considerado necesarias valorando las implicaciones que conllevan. Este trabajo, al final, no deja de ser un PoC (Proof of Concept), diseñado para mostrar cómo se implementarían las diferentes tecnologías.

Por ello, en la búsqueda de un mejor rendimiento, se ha optado por utilizar HTTP en lugar de HTTPS tanto para la aplicación web como para la API que permite la conexión con los nodos de OrbitDB, siendo conscientes de que esta decisión no es la más recomendable y que representa un riesgo en términos de seguridad. Sin embargo, este riesgo es fácilmente mitigable realizando los cambios apropiados en cada uno de los servidores, forzando en ellos el uso de HTTPS y prohibiendo el uso de HTTP.

Al tomar esta decisión, hemos priorizado la facilidad para realizar pruebas y cambios en el sistema. Implementar HTTPS en una etapa temprana del desarrollo podría haber complicado estas tareas, por lo que decidimos utilizar HTTP temporalmente. En el caso de utilizar este sistema en un entorno real, la transición a HTTPS se llevará a cabo de manera sencilla, mejorando así la seguridad del sistema sin afectar significativamente el rendimiento ni la flexibilidad durante el desarrollo.

De igual manera, no se ha llevado a cabo un análisis exhaustivo de la seguridad de la aplicación web. En el futuro, será fundamental realizar una revisión minuciosa de la seguridad de cada una de las librerías utilizadas, así como de cada uno de los componentes de la web, para evitar posibles ataques que puedan comprometer el sistema.

6.1.2 OrbitDB

OrbitDB permite definir controladores de acceso para cada base de datos. De esta forma, se puede especificar quién está autorizado para leer o escribir datos en la base de datos. Esto limita la capacidad de los nodos no legítimos para realizar modificaciones no autorizadas.

Los controladores de acceso determinan los permisos de escritura que tiene un usuario en una base de datos. Por defecto, el acceso de escritura está restringido al usuario que creó la base de datos. Los controladores de acceso permiten ampliar estos permisos a otros usuarios, además del creador de la base de datos.

Al abrir una base de datos por primera vez se pasará un controlador de acceso. Una vez creada, el acceso de escritura a la base de datos estará restringido únicamente a los usuarios especificados en la lista de control de acceso. Por defecto, solo el usuario que crea la base de datos tendrá permisos de escritura.

Para complementar el sistema de control de acceso, OrbitDB utiliza identidades para asegurar que solo los usuarios autorizados puedan realizar ciertas acciones, como escribir o modificar datos en la base de datos. Estas identidades se generan mediante pares de claves criptográficas (clave pública y clave privada), utilizando algoritmos de firma digital basados en criptografía de curva elíptica, especialmente el algoritmo de firma digital ECDSA (Elliptic Curve Digital Signature Algorithm), debido a su eficiencia y seguridad en entornos distribuidos y de baja latencia como IPFS. Con el poder de cómputo actual, el algoritmo ECDSA no es susceptible a ataques. Sin embargo, en un escenario post-cuántico, donde las computadoras cuánticas sean capaces de romper la criptografía de curva elíptica, ECDSA no sería seguro. Esto significa que las identidades y las firmas en OrbitDB podrían ser forjadas, comprometiendo la integridad y autenticidad de los datos.

Cuando un usuario realiza una transacción, la firma digital asegura la autenticidad de la acción, y otros nodos en la red pueden verificarla usando la clave pública del usuario. Esto garantiza que solo los usuarios legítimos puedan interactuar con la

base de datos, proporcionando una capa adicional de seguridad y confianza en el sistema.

En resumen, los controladores de acceso y las identidades trabajan conjuntamente para asegurar que solo los usuarios autorizados puedan modificar los datos en OrbitDB. Los controladores de acceso determinan quién tiene permisos de escritura, mientras que las identidades aseguran que cada transacción es realizada por un usuario legítimo y verificable. Esta combinación de mecanismos de seguridad fortalece la integridad y confiabilidad del sistema.

OrbitDB se basa en una red de pares (peers) donde la confianza se distribuye entre múltiples nodos. En un entorno bien diseñado, los nodos legítimos superan en número a los nodos maliciosos. Si un nodo intenta propagar datos falsificados, los demás nodos de la red pueden rechazar estos datos mediante un mecanismo de consenso, asegurando que solo los datos válidos sean aceptados y replicados.

OrbitDB mantiene un historial de versiones de cada base de datos, lo que permite auditar y verificar todas las modificaciones realizadas. Esto significa que cualquier intento de manipulación puede ser detectado y rastreado. Si un nodo no legítimo intenta modificar datos históricos, los cambios serán evidentes y podrán ser revertidos utilizando las versiones anteriores de los datos.

6.1.3 Oráculo

Actualmente el sistema se centra en un único oráculo, que es responsable de proporcionar datos externos a la blockchain. Esta centralización puede ser una preocupación, ya que cualquier fallo o mal funcionamiento del oráculo puede afectar al correcto funcionamiento del sistema de votación.

Sin embargo, el diseño propuesto está pensado para poder descentralizar el oráculo, incluyendo varios nodos que trabajarán en paralelo, aumentando la disponibilidad y la seguridad del sistema. Además, la distribución de funciones entre varios nodos permite una mayor eficiencia en el procesamiento de datos. Cada nodo puede trabajar en paralelo, lo que acelera el tiempo de respuesta del oráculo.

La incorporación de varios nodos para la función del oráculo aumentará significativamente la seguridad del sistema. En lugar de depender de un único punto de fallo, un oráculo descentralizado distribuye las funciones entre múltiples nodos, reduciendo el riesgo de que un solo punto de fallo pueda comprometer todo el sistema. Esta descentralización también conlleva varios desafíos. Uno de los problemas principales es la coordinación y consenso entre nodos.

Aunque los oráculos descentralizados buscan minimizar la necesidad de confianza centralizada, es importante entender que no eliminan completamente la necesidad de confianza. En lugar de depender de una única fuente confiable, los oráculos descentralizados distribuyen la confianza entre múltiples participantes [Ben20]. Cada nodo en el oráculo contribuye con su propia verificación y suministro de datos. La idea es que, al agregar las respuestas de muchos nodos, se reduce la posibilidad de que datos incorrectos o maliciosos sean aceptados.

En resumen, aunque nuestra implementación inicial se basa en un oráculo centralizado, el enfoque propuesto para un oráculo descentralizado ofrece múltiples ventajas, incluyendo una mayor seguridad, eficiencia, redundancia y descentralización de la confianza. No obstante, también conlleva desafíos significativos, como gestionar la coordinación entre nodos.

Descentralización del oráculo

En primer lugar, para descentralizar los oráculos es necesario definir el protocolo de consenso que usarán los nodos para ponerse de acuerdo sobre los datos proporcionados. El proceso para llegar a un consenso es el siguiente:

Cada nodo debe monitorizar la blockchain en espera de los eventos emitidos por el contrato del oráculo. Cada nodo detectará el evento, obtendrá el archivo y lo subirá a IPFS. Como se comentó previamente en la sección 4.2, IPFS genera un identificador conocido como CID que es una representación única del contenido del archivo y a la vez funciona como su dirección. Este se genera utilizando una función hash criptográfica sobre el contenido del archivo y produce un valor único para un conjunto de datos específico. Por lo tanto, mientras el contenido del archivo sea exactamente igual, el hash generado será el mismo independientemente del nodo desde el cual se haya subido. Debido a que IPFS es un sistema basado en contenido, cualquier archivo con el mismo contenido tendrá el mismo CID en toda la red IPFS. En un sistema ideal, todos los nodos deberían obtener el mismo hash si el archivo y el proceso de subida son idénticos.

Cada nodo registra el hash de IPFS obtenido después de subir el archivo y los comparan. Si todos los nodos obtuvieron el mismo hash, se puede asumir que el archivo ha sido subido correctamente y de manera consistente. Los nodos pueden usar un mecanismo de votación simple en el cual si la mayoría (más del 50%) de los nodos reporta el mismo hash, se considera que este hash es válido. Sin embargo, por seguridad, es mejor utilizar algoritmos de consenso más sofisticados que no se basen

únicamente en la mayoría simple, sino que también consideren la reputación y el historial de los nodos.

Al utilizar un sistema de reputación, la confianza no se basará únicamente en la mayoría simple, sino la calidad y el historial de los nodos, asignándole un peso en el consenso a cada nodo en base a la confianza depositada sobre él y el hash aportado.

Asimismo, si el nodo con mayor reputación detecta un comportamiento anómalo, puede indicar aquellos nodos que están actuando de manera maliciosa y el sistema debería identificar y penalizar a estos nodos por su actuación negligente. De esta forma se podría detectar un ataque del 51%, un ataque que implica que un atacante o un grupo de atacantes obtengan el control de más del 50% de los nodos de oráculo, permitiéndoles manipular los datos proporcionados a los contratos inteligentes. Este tipo de ataque puede comprometer gravemente la integridad y la fiabilidad de la red.

Para evitar que todos los nodos intenten registrar el hash en la blockchain simultáneamente, se puede seleccionar un nodo coordinador. Este nodo puede ser elegido de varias maneras:

- Rotación: Los nodos se turnan para ser el coordinador.
- Elección aleatoria: Se selecciona un nodo al azar.
- Elección basada en mérito: Se elige el nodo con mayor reputación.

La elección basada en mérito es generalmente la más segura porque selecciona el nodo coordinador en función de su historial de confiabilidad y desempeño, asegurando que el nodo más adecuado y seguro asuma el rol de coordinador en cada ciclo.

Una vez llegado a un consenso sobre el hash del archivo, el nodo coordinador enviará una transacción al contrato del oráculo en la blockchain con el hash de IPFS acordado. Este hash es entonces accesible para el contrato externo que originalmente solicitó la subida del archivo.

El flujo de datos sería entonces de la siguiente manera:

- Evento detectado: Todos los nodos detectan un evento emitido por un contrato externo que solicita la subida de un archivo a IPFS.
- Obtención del archivo: Cada nodo obtiene el archivo a través del evento.
- Subida a IPFS: Los nodos suben el archivo a IPFS y obtienen un hash.

- Registro y comparación de hashes: Los nodos registran sus hashes y los comparan.
- Consenso mayoritario: Los nodos llegan a un consenso sobre el hash del archivo subido a la red IPFS. Los nodos que proporcionaron el hash incorrecto reciben una penalización en su reputación.
- Selección del coordinador: Se selecciona un nodo coordinador en base a su historial de confiabilidad y desempeño.
- Envío de transacción: El nodo coordinador envía una transacción al contrato del oráculo con el hash.
- Actualización del contrato: El contrato del oráculo actualiza su estado con el hash.
- Verificación por el contrato externo: El contrato externo verifica que el hash ha sido actualizado correctamente en el contrato del oráculo.

6.2 Posibles ataques en redes blockchain

En esta sección se abordarán diversos tipos de ataques que pueden comprometer la integridad y seguridad de la blockchain.

6.2.1 Ataque del 51%

El ataque del 51% [SM19] representa una de las amenazas más significativas para la integridad y seguridad de las redes blockchain. Este ataque se produce cuando un solo actor o un grupo de atacantes logra controlar más del 50% del poder de procesamiento (hashrate) de la red. Al alcanzar esta mayoría, los atacantes obtienen la capacidad de monopolizar la generación de bloques y de realizar cambios potencialmente dañinos en la blockchain. Cuanto mayor sea el poder de hash total de la red blockchain, más costoso se vuelve el ataque. Por lo tanto, se supone que las criptomonedas con un alto poder de hash en la red son más seguras contra el ataque del 51%.

La seguridad de este sistema depende de la distribución descentralizada del poder de hash entre los participantes de la red. Sin embargo, si un atacante o grupo de atacantes logra controlar el 51% o más del poder de hash pueden monopolizar la minería, garantizando que sean los únicos en resolver los problemas criptográficos

y por ende las recompensas asociadas, o llevar a cabo un ataque de doble gasto, en el que realizan transacciones mientras simultáneamente minan en una cadena alternativa en privado y una vez que la cadena alternativa es lo suficientemente larga, la presentan a la red, invalidando las transacciones previas y permitiéndoles gastar las mismas monedas nuevamente.

Para mitigar el riesgo de un ataque del 51%, existen varias estrategias. A continuación se explican varias de ellas:

- Cambiar el algoritmo de consenso a algoritmos alternativos, como la prueba de participación (Proof of Stake, PoS), que no dependen del poder de hash y pueden ofrecer diferentes mecanismos de seguridad.
- Implementar un sistema para penalizar los bloques retrasados, aumentando los costos para los atacantes y notificando a la red sobre bifurcaciones, limitando así las transacciones fraudulentas [Gar+18].
- Delayed proof of work (dPoW), una técnica que emplea una cadena PoW asignada para guardar transacciones, permitiendo a la cadena defenderse de ataques del 51%. No sigue la regla de la cadena más larga y utiliza nodos notariales para verificar la seguridad del hash.
- La minería combinada permite minar múltiples criptomonedas simultáneamente, beneficiando a criptomonedas con bajo poder de hash al utilizar la potencia de otras con mayor hash. Aunque no es una técnica de seguridad per se, ayuda a mitigar el ataque del 51%.

6.2.2 Ataques MitM

Los ataques Man-in-the-Middle (MitM) son un tipo de ataque en el que un atacante intercepta y altera las comunicaciones entre dos partes sin que estas se den cuenta. Un ataque MitM ocurre cuando un atacante se posiciona secretamente entre dos partes que se comunican entre sí y manipula o espía la comunicación.

En el contexto de las redes blockchain, los ataques MitM representan una amenaza significativa, ya que permiten a un atacante interceptar el tráfico de red entre nodos de la blockchain, alterarlo antes de reenviarlo o incluso hacerse pasar por uno de los nodos de la red, engañando a los otros nodos para que piensen que están comunicándose con una entidad legítima cuando en realidad están enviando información al atacante.

Existen dos métodos notables para llevar a cabo estos ataques: el secuestro de rutas BGP (Border Gateway Protocol) y los ataques DNS (Domain Name System).

El secuestro de rutas BGP es un tipo específico de ataque MitM donde el atacante altera las rutas de tráfico de la red, redirigiendo el tráfico destinado a un nodo legítimo hacia su propio nodo malicioso. Esto se logra manipulando las rutas BGP, que son esenciales para la correcta dirección del tráfico. Al interceptar este tráfico, el atacante puede acceder a datos sensibles, alterar transacciones o incluso desestabilizar el consenso de la red blockchain. Este método ha sido utilizado con éxito en el pasado, como en el caso de un ataque en 2014 que resultó en el robo de 83,000 dólares [EGJ18].

Los ataques DNS, por otro lado, implican la manipulación del sistema de nombres de dominio (DNS) para redirigir el tráfico destinado a un servicio legítimo hacia un servidor controlado por el atacante. Este tipo de ataque es especialmente peligroso en el contexto de las billeteras de criptomonedas en línea. Al comprometer el DNS, el atacante puede redirigir a los usuarios a un sitio web falso que parece idéntico al original, capturando así las claves privadas y credenciales de los usuarios [18a].

Los ataques de ARP-spoofing también son efectivos y funcionarían de forma similar a los ataques por secuestro de BGP, pero su riesgo puede mitigarse fácilmente detectando y restringiendo el número de direcciones MAC por puerto de red, además el nodo atacante debe asumir la carga de todo el tráfico de comunicación entre los mineros en la red blockchain, lo que genera una sobrecarga adicional [EGJ18].

Para mitigar el riesgo de un ataque MitM, existen varias estrategias. A continuación se explican varias de ellas:

- Implementar mecanismos de autenticación mutua entre nodos para garantizar que los nodos se identifiquen de manera segura antes de intercambiar información.
- Implementar RPKI (Resource Public Key Infrastructure) puede ayudar a validar los anuncios de rutas BGP y prevenir que los atacantes manipulen las rutas de tráfico. Además de monitorizar continuamente las rutas BGP para detectar y responder a cambios inesperados que podrían indicar un ataque.
- Para prevenir ataques DNS Spoofing, se puede utilizar DNSSEC (Domain Name System Security Extensions) para añadir una capa de seguridad al sistema DNS.
- El ARP-spoofing puede ser mitigado utilizando protecciones en switches de red.

- Utilizar una billetera de criptomonedas como MetaMask puede proteger contra sitios web falsos, ya que cuenta con medidas para detectar dominios de phishing dirigidos a usuarios de Web3.

6.2.3 Ataques cuánticos

Con los recientes avances en la computación cuántica, los ataques cuánticos se han convertido en una nueva amenaza para la seguridad de la blockchain. Estos ataques aprovechan la capacidad de los ordenadores cuánticos para resolver problemas matemáticos complejos mucho más rápido que los ordenadores clásicos. Esto podría comprometer la criptografía que actualmente protege las transacciones y datos en las redes blockchain.

En Ethereum, la seguridad de las transacciones y la gestión de contratos inteligentes dependen en gran medida de la criptografía de clave pública, como el algoritmo de curva elíptica secp256k1, utilizado para crear claves públicas y privadas, y para firmar y verificar transacciones en la mayoría de las criptomonedas. La clave privada, que es esencial para firmar transacciones y contratos inteligentes, está protegida por criptografía que, en la actualidad, es segura contra ataques de ordenadores clásicos.

Los ataques cuánticos pueden romper la seguridad de secp256k1 de las siguientes maneras:

- Algoritmo de Shor [Sho97]: Este es un algoritmo cuántico que puede resolver problemas matemáticos que actualmente se consideran difíciles para los ordenadores clásicos. En particular, el algoritmo de Shor puede factorizar grandes números primos y resolver el problema del logaritmo discreto en tiempo polinómico. Dado que secp256k1 se basa en el problema del logaritmo discreto, el algoritmo de Shor puede, en teoría, calcular la clave privada a partir de la clave pública en un tiempo mucho más corto que el requerido por los métodos clásicos.
- Algoritmo de Grover [Gro96]: Aunque no afecta directamente al secp256k1, el algoritmo de Grover podría ser utilizado para acelerar la búsqueda de claves privadas, reduciendo a la mitad el número de intentos necesarios para una búsqueda de fuerza bruta. El algoritmo de Grover puede acelerar la generación de hashes, lo que permite recrear toda la cadena de bloques. Además, el algoritmo de Grover puede adaptarse para detectar colisiones de hash, que

pueden usarse para reemplazar bloques de una cadena de bloques preservando al mismo tiempo su integridad [FF20].

En el caso del fichero que se sube cifrado a IPFS con los datos de los votos emitidos, se ha utilizado AES-128 para cifrarlo. AES-128 es resistente contra ataques cuánticos en la práctica, pero AES-256 ofrece una seguridad más robusta frente a ataques cuánticos a largo plazo. Para un nivel adicional de seguridad contra ataques cuánticos, se podría pasar a utilizar AES-256 en lugar de AES-128.

Para AES-128, Grover reduciría el tiempo necesario para romper el cifrado a aproximadamente 2^{64} intentos en lugar de 2^{128} , lo cual sigue siendo una cantidad computacionalmente inabordable con los recursos actuales. Sin embargo, para AES-256, Grover solo reduce el esfuerzo a 2^{128} , lo cual ofrece mayor seguridad.

Conclusiones

Blockchain se ha convertido en una tecnología revolucionaria con el potencial de transformar la ciberseguridad de manera significativa. El desarrollo de un sistema de votación electrónica basado en Ethereum ha demostrado el gran potencial que tiene la tecnología blockchain para mejorar la seguridad, transparencia y eficiencia de los procesos electorales. Los contratos inteligentes escritos en Solidity permiten la automatización de las reglas de votación y garantizan que cada voto se registre de manera inmutable y verificable. La descentralización de la blockchain elimina la necesidad de una autoridad central, reduciendo el riesgo de fraude y manipulación.

La integración de blockchain con tecnologías de almacenamiento descentralizado como IPFS y OrbitDB potencia la creación de sistemas más seguros y resistentes, que no dependan de infraestructuras centralizadas. Esto, a su vez, mejora la seguridad al reducir los puntos únicos de vulnerabilidad y aumentar la resistencia de los sistemas contra ataques.

La descentralización del sistema también nos permite escalarlo de acuerdo con las circunstancias y necesidades específicas, añadiendo o eliminando nodos según lo requiera el entorno. Esta flexibilidad es una de las principales ventajas de los sistemas descentralizados, ya que permite ajustar la capacidad y rendimiento del sistema de manera dinámica. Al aumentar el número de nodos, se mejora la seguridad y resistencia del sistema, ya que se reduce el riesgo de que un ataque o falla afecte a toda la red. Cada nodo adicional contribuye a la redundancia de los datos y a la distribución de la carga de trabajo, haciendo más difícil para un atacante comprometer el sistema.

Sin embargo, es importante tener en cuenta que el aumento en el número de nodos también incrementa la complejidad y la carga de trabajo del sistema. Cada nodo debe procesar y almacenar datos, lo que puede aumentar el tiempo de procesamiento y los requisitos de recursos. Además, más nodos implican más comunicaciones entre ellos, lo que puede afectar el rendimiento de la red en términos de latencia y ancho de banda. Por lo tanto, es crucial encontrar un equilibrio óptimo entre la seguridad y el rendimiento del sistema, ajustando el número de nodos según las necesidades operativas y la capacidad de la infraestructura.

7.1 Limitaciones y trabajo futuro

7.1.1 Smart Contract

Como parte del trabajo futuro para mejorar este sistema de votación, una de las principales tareas será modificar el contrato inteligente para permitir diferentes opciones de votación. En este trabajo, por simplicidad, se ha optado por utilizar una opción fija con solo dos alternativas, ya que no era el enfoque principal del sistema. En lugar de tener solo dos opciones de votación definidas estáticamente, se podría utilizar un array para almacenar múltiples opciones y un mapping para registrar los votos de cada opción.

Otra de las tareas pendientes es realizar un análisis del rendimiento de la red. El rendimiento en las redes blockchain es crucial, ya que el número de votantes puede afectar significativamente al funcionamiento del contrato inteligente. Por lo tanto, sería interesante en el futuro llevar a cabo una medición detallada para evaluar cómo el tráfico impacta en el rendimiento del sistema. Esto permitirá identificar posibles cuellos de botella y optimizar el contrato inteligente para manejar un mayor volumen de transacciones de manera eficiente.

7.1.2 Oráculo

Actualmente, el diseño del oráculo está limitado y solo se puede recuperar la información del último voto emitido. Para cambiar esto, sería necesario modificar el oráculo, lo cual conllevaría un peor rendimiento por parte de este, o bien modificar el sistema para que cada votación tenga su propio oráculo dedicado.

La opción de modificar el oráculo implicaría ajustar su funcionalidad para gestionar y almacenar múltiples votos de manera concurrente, lo cual podría impactar negativamente en su rendimiento debido a la carga adicional y la gestión de datos más compleja.

Por otro lado, asignar un oráculo dedicado a cada votación podría proporcionar una solución más escalable y eficiente en términos de rendimiento. Cada oráculo estaría específicamente diseñado para gestionar los votos de una sola votación, evitando así la necesidad de gestionar múltiples votos dentro de un mismo contexto y mejorando la capacidad de recuperación de datos históricos de votación.

Como se ha explicado en la sección 6.1, la descentralización del oráculo, aunque ha sido diseñada, no ha sido implementada en el sistema. La descentralización del oráculo nos permitiría utilizar diferentes oráculos e incluso permitir que cada una de las partes interesadas en la votación, como los posibles candidatos, partidos políticos y organizaciones que se presenten a las elecciones, puedan desplegar sus propios nodos y de esta forma participar en la red. Esto mejoraría la seguridad, la confianza y el rendimiento de la red, permitiendo una mayor transparencia y resistencia a fallos o manipulaciones. Así, implementar esta característica significaría un avance en la robustez y fiabilidad del sistema de votación.

Además, en el futuro se podrían implementar nuevas funciones en los oráculos, lo que permitiría una conexión sin límites entre la blockchain y el mundo off-chain. Esto ampliaría significativamente las posibilidades y aplicaciones del sistema, facilitando la automatización y la ejecución de tareas complejas que dependan de datos externos.

7.1.3 OrbitDB

Al igual que con la descentralización del oráculo, la implementación de más nodos de OrbitDB nos permitiría mejorar la seguridad y el rendimiento de la red. Sin embargo, en este caso, sería necesario realizar un mayor número de cambios en comparación con la descentralización del oráculo, ya que se requiere configurar las direcciones de cada uno de estos nodos. La redundancia de datos proporcionada por múltiples nodos aumentaría la resistencia frente a ataques de denegación de servicio y a intentos de manipulación de datos por parte de diferentes atacantes. Además, esta configuración podría mejorar la disponibilidad y la integridad de los datos almacenados en la red, asegurando que la información sea accesible incluso en caso de fallos o ataques a algunos de los nodos.

Además, el acceso a la API que permite interactuar con los nodos de OrbitDB podría securizarse utilizando una prueba de conocimiento cero (ZKP, Zero-Knowledge Proof), en el que el usuario firmará, utilizando MetaMask, un mensaje generado aleatoriamente. En base a este mensaje firmado se comprobará la autenticidad de la firma y se comprobará si la cuenta que ha firmado el mensaje corresponde a una de las cuentas autorizadas por los administradores del sistema. Al utilizar las firmas digitales como forma de autenticación se garantiza un alto nivel de seguridad ya que, las claves privadas necesarias para firmar mensajes están protegidas y nunca se comparten, lo que reduce significativamente el riesgo de compromiso de la cuenta. Además, aseguran que únicamente se conozca la dirección de la cartera asociada al

usuario, un dato que es público, pero que no tiene por qué ser asociado a ningún dato del usuario que permita revelar su identidad.

Las ZKP ofrecen muchas ventajas a nivel de seguridad:

- Permiten verificar la autenticidad de la información sin revelar la información en sí misma. En este contexto, permite al sistema verificar que un usuario está autorizado sin que el usuario tenga que revelar su clave privada o cualquier otra información sensible. Esto protege la privacidad del usuario al máximo.
- Dado que las claves privadas nunca se transmiten ni se comparten, el riesgo de que estas sean interceptadas por un tercero malintencionado es casi nulo. Esto significa que, incluso si la comunicación entre el usuario y el sistema fuese interceptada, el atacante no podría obtener suficiente información para comprometer la cuenta del usuario.
- La naturaleza aleatoria del mensaje que se firma impide que un atacante pueda reutilizar un mensaje firmado anterior para intentar autenticarse. Cada intento de autenticación requiere un nuevo mensaje aleatorio, lo que neutraliza los ataques de repetición (replay attacks).
- Al eliminar la necesidad de compartir información sensible, las ZKP reducen significativamente el riesgo de ataques de ingeniería social donde un atacante podría engañar a un usuario para que revele información confidencial.

7.1.4 Front-End

En un principio, la intención era almacenar en IPFS los datos de los votos emitidos por los usuarios, con la finalidad de proporcionarles un comprobante que garantizara la correcta emisión de su voto. Al decidir cifrar los datos antes de subirlos a la red como medida de seguridad adicional ha surgido un desafío: debido a las restricciones de tiempo, no se ha implementado una solución para recuperar los archivos cifrados desde el front-end. Es importante notar que esto solo afecta al front-end, puesto que el oráculo sigue completamente funcional, y puede comprobarse su funcionamiento si se realizan las llamadas al contrato del oráculo desde una herramienta como Remix IDE.

Como se explicó en la sección 6.1.1, es necesario realizar una securización de la aplicación web, lo que implica forzar el uso de HTTPS, tarea que no debería llevar mucho tiempo ni ser compleja, pero se ha omitido inicialmente para facilitar las pruebas y el desarrollo.

De igual manera, es imprescindible llevar a cabo un análisis exhaustivo de la seguridad de la aplicación. Esto incluye realizar pruebas de penetración para identificar y mitigar posibles puntos débiles. Estas pruebas permitirán descubrir vulnerabilidades y adoptar las medidas necesarias para fortalecer la seguridad del sistema.

En el caso de que esta aplicación llegue a usarse en un entorno real, será necesario establecer un proceso de actualización continuo. Este proceso debe asegurar que las librerías y componentes se mantengan al día con las últimas correcciones de seguridad y mejoras.

En resumen, aunque en esta fase inicial no se ha profundizado en el análisis de seguridad de la aplicación web, es imperativo que en etapas posteriores se dediquen los recursos necesarios para garantizar que la aplicación sea robusta y resistente frente a posibles amenazas. Esto asegurará que la aplicación no solo funcione correctamente, sino que también esté protegida contra posibles ataques y vulnerabilidades.

7.1.5 Blockchain

A pesar de que en este trabajo se ha utilizado Hardhat como red blockchain, en el futuro sería necesario realizar una evaluación tanto de seguridad como de rendimiento de la red. Usando una blockchain privada de Ethereum, es posible enviar cientos de transacciones por segundo a la cadena de bloques [Hja+18], pero sería interesante desarrollar medidas adicionales para conseguir un mayor rendimiento.

Esto incluiría realizar una comparación detallada entre diferentes redes privadas y públicas para evaluar su desempeño en términos de latencia, velocidad de transacción y eficiencia general, lo que ayudará a determinar cuál es la mejor opción para el caso de uso específico basado en los requerimientos de rendimiento. Además, sería interesante medir el rendimiento de la blockchain en función del número de transacciones por segundo (TPS) que puede manejar, una métrica crucial para evaluar la capacidad de la red para escalar y manejar un alto volumen de transacciones sin degradar la calidad del servicio.

7.1.6 Securización post-cuántica

Tal y como se explicó en la sección 6.2.3, con los recientes avances en la computación cuántica, los ataques cuánticos se han convertido en una nueva amenaza para la seguridad de la blockchain.

A pesar de que en la actualidad no existen ordenadores cuánticos con la capacidad suficiente para llevar a cabo los ataques mencionados, se espera que en los próximos años estos ordenadores cuánticos sean capaces de romper fácilmente los sistemas criptográficos actuales. Por esta razón, sería prudente realizar un análisis de seguridad post-cuántico en el sistema e ir adoptando algoritmos criptográficos resistentes a la computación cuántica a medida que avanzan las investigaciones en este campo.

Anexo - Planificación

A.1 Planificación temporal

A continuación, se detalla la planificación que se llevó a cabo para la realización de este trabajo, así como los distintos ciclos y fases en las que se dividió el trabajo. Se puede ver de forma gráfica en la Fig. A.1.

A.1.1 Ciclo 1 - Planificación inicial

Este ciclo se corresponde con los capítulos 1 y 2 de la memoria y está compuesta por las siguientes fases:

- Fase 1 - Análisis y revisión del estado del arte. Definición de requisitos: Documentar los requisitos funcionales y no funcionales de la aplicación.
 - Duración: 04/03/2024 al 15/03/2024.
- Fase 2 - Identificación de componentes: Listar todos los componentes necesarios (smart contract, oráculo, nodos de OrbitDB, front-end) y sus respectivas funciones.
 - Duración: 15/03/2024 al 29/03/2024.

A.1.2 Ciclo 2 - Diseño y desarrollo de los componentes independientes

Este ciclo se corresponde con los capítulos 3 y 4 de la memoria y está compuesta por las siguientes fases:

- Fase 1 - Diseño de los componentes independientes.
 - Duración: 29/03/2024 al 04/04/2024.
- Fase 2 - Desarrollo de los componentes independientes.

- Duración: 04/04/2024 al 10/05/2024.

A.1.3 Ciclo 3 - Integración y pruebas de integración

Este ciclo se corresponde con el capítulo 5 de la memoria y está compuesta por las siguientes fases:

- Fase 1 - Pruebas unitarias: Realizar pruebas unitarias en cada componente para asegurarse de que funcionan correctamente de manera aislada.
 - Duración: 10/05/2024 al 20/05/2024.
- Fase 2 - Pruebas del sistema: Realizar pruebas completas de la aplicación en su totalidad.
 - Duración: 20/05/2024 al 30/05/2024.
- Fase 3 - Validación de requisitos: Verificar que todos los requisitos definidos inicialmente se cumplen.
 - Duración: 30/05/2024 al 31/05/2024.

A.1.4 Ciclo 4 - Análisis del sistema

Este ciclo se corresponde con el capítulo 6 de la memoria y está compuesta por las siguientes fases:

- Fase 1 - Evaluación de la seguridad del sistema.
 - Duración: 31/05/2024 al 06/06/2024.
- Fase 2 - Análisis de los posibles ataques al sistema.
 - Duración: 06/06/2024 al 13/06/2024.

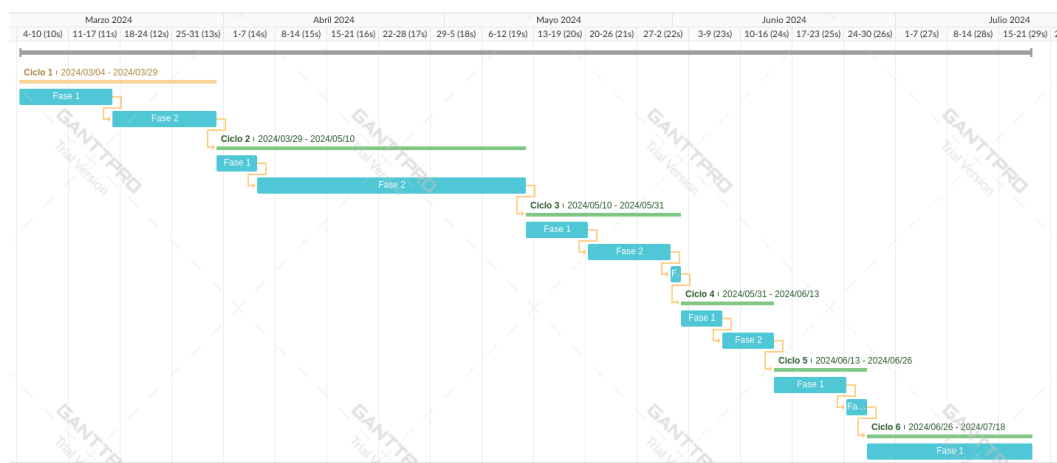


Figura A.1.: Diagrama de Gantt de la planificación.

A.1.5 Ciclo 5 - Despliegue

Este ciclo se corresponde con el anexo B de la memoria y está compuesta por las siguientes fases:

- Fase 1 - Preparación para el despliegue: Asegurarse de que todos los componentes están listos para desplegarse.
 - Duración: 13/06/2024 al 24/06/2024.
- Fase 2 - Despliegue: Desplegar la aplicación en un entorno de pruebas.
 - Duración: 24/06/2024 al 26/06/2024.

A.1.6 Ciclo 6 - Realización de la memoria

Este ciclo se corresponde con la documentación de la memoria presentada con este trabajo.

- Fase 1 - Documentación de la memoria.
 - Duración: 26/05/2024 al 18/07/2024.

Puesto	Precio en /hora	Total de horas	Coste
Tutor académico (1)	30	50	1500
Tutor académico (2)	30	50	1500
Desarrollador	20	406	8120
			Total 11120

Cuadro A.1.: Tabla de los recursos humanos y sus costes.

A.2 Costes estimados

En esta sección, se tratan de estimar los salarios de los recursos humanos necesarios para este Trabajo de Fin de Máster. En la tabla A.1 se puede ver esta información.

Anexo - Instalación y walkthrough

El código del sistema desarrollado se encuentra en el repositorio de GitHub disponible en [24n], bajo la licencia GPL v3 [Fou07].

B.1 Instalación

B.1.1 Hardhat

En primer lugar después de instalar Hardhat, es necesario desplegar una red con el siguiente comando:

```
1 npx hardhat node --port 8544
```

Esto iniciará la red blockchain que usaremos en el proyecto y creará 10 cuentas que podremos importar a MetaMask y utilizar en el sistema de votación. Para este ejemplo, utilizaremos las cuentas 18 y 19, mostradas en la Fig. B.1.

```
Account #18: 0xdD2FD4581271e230360230F9337D5c0430Bf44C0 (10000 ETH)
Private Key: 0xde9be858da4a475276426320d5e9262ecfc3ba460bfac56360bfa6c4c28b4ee0

Account #19: 0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199 (10000 ETH)
Private Key: 0xdf57089febbacf7ba0bc227dafbffa9fc08a93fdc68e1e42411a14efcf23656e

WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.
```

Figura B.1.: Cuentas a importar.

Para utilizar MetaMask con esta red, tendremos que añadirla manualmente a la configuración de MetaMask. Para ello, tendremos que ir a Configuración, Redes, y añadir manualmente cada uno de los campos que se muestran en la Fig. B.2.

Una vez configurada la red, es necesario importar las cuentas en Metamask. Para ello, es necesario seguir el proceso mostrado en las Fig. B.3 y B.4, utilizando la clave privada de la cuenta para importarla.

Nombre de la red

Hardhat 8544

Nueva dirección URL de RPC

http://127.0.0.1:8544/

Identificador de cadena ⓘ

31337

En este momento, la red Hardhat está utilizando este identificador de cadena.

Símbolo de moneda

HardhatEther

Suggested ticker symbol: GO

This token symbol doesn't match the network name or chain ID entered. Many popular tokens use similar symbols, which scammers can use to trick you into sending them a more valuable token in return. Verify everything before you continue.

Dirección URL del explorador de bloques (Opcional)

Cancelar Guardar

Figura B.2.: Captura de los campos a añadir para configurar la red.

< Añadir cuenta X

+ Add a new account

🔗 Importar cuenta

🏠 Agregar monedero físico

Figura B.3.: Importar la cuenta de Hardhat en Metamask.

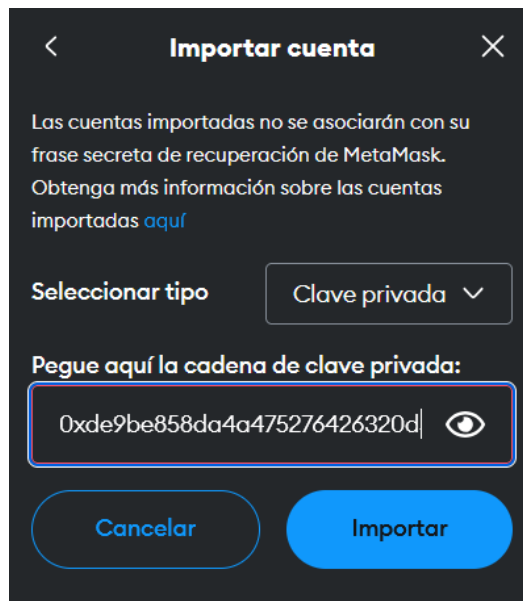


Figura B.4.: Introducir la clave privada de la cuenta para importarla.

Una vez completado el paso anterior, ya tendremos importada la cuenta en MetaMask, y se nos mostrará la información sobre ella que se puede apreciar en la Fig. B.5.

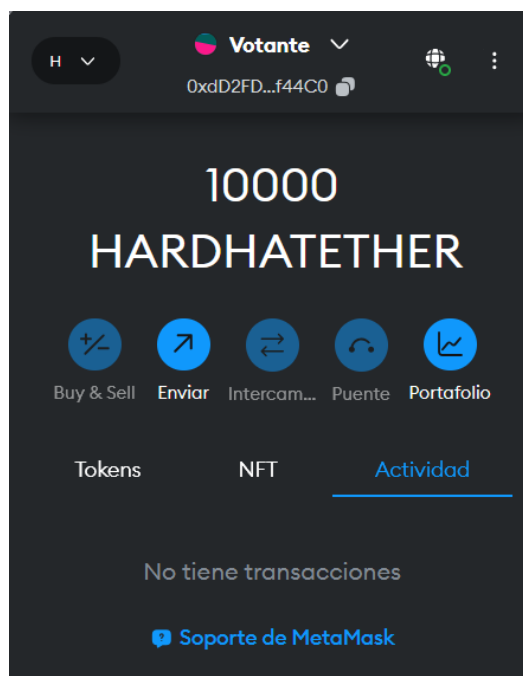


Figura B.5.: Cuenta importada en MetaMask.

Al entrar por primera vez en la aplicación web, recibiremos un mensaje en MetaMask que nos informará de que la cuenta importada no está conectada al sitio web actual, como se muestra en la Fig. B.6. Una vez aceptemos la conexión, ya podremos utilizar la cuenta con la aplicación.

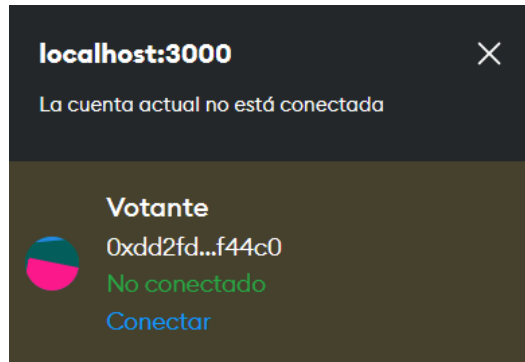


Figura B.6.: Conectar la cuenta a la aplicación web utilizando MetaMask.

B.1.2 OrbitDB

Para levantar el nodo de OrbitDB, en primer lugar es necesario asegurarse de que Node.js y npm se encuentran instalados en el equipo. Para instalar las dependencias del proyecto, se puede ejecutar el siguiente comando en el directorio raíz (orbitdb-storage):

```
1 npm install
```

Para levantar el nodo de OrbitDB, es suficiente con ejecutar el siguiente comando:

```
1 node server.js
```

Esto levantará un nodo de OrbitDB con la API en el puerto 3001 como se muestra en la Fig. B.7. La aplicación ya está configurada para utilizar este puerto para añadir y recuperar la lista de las votaciones abiertas. En el caso de querer utilizar más nodos, es necesario realizar cambios como se ha indicado en la sección 4.3.

```
PS C:\Users\pedro\OneDrive\Escritorio\TFM Munics\Código\orbitdb-storage> node server.js
my-db address /orbitdb/zdpuAkogv7a8jwjmdBk8U6iJg65c2Z7QJt7Rf7CyEM8t4uSGi
Server running at http://localhost:3001
Nuevo registro añadido: { key: "0x73511669fd4dE447feD18BB79bAFeAC93aB7F31f", value: "Biparti
to", hash: "zdpuAtgpYf1yLMu18GYDdsPX7wqfoXynjuNdvQgoh1msWrapg" }
█
```

Figura B.7.: Ejemplo de la ejecución del nodo de OrbitDB.


```

PS C:\Users\pedro\OneDrive\Escritorio\TFM Muncis\Código\oráculo> docker run --name nombre_contenedor -p 8000:8000 nombre_imagen
generating ED25519 keypair...done
peer identity: 12D3KookDadu2Ar5s1Qeb6PFdEpTVQKgpXHU91ooVfr47Jah2w5LR
initializing IPFS node at /root/.ipfs
removed /dnsaddr/bootstrap.libp2p.io/p2p/QmNnooDu7bfjPFoTZYxMNLWUQJyrVwtbZg5gBMjTeZGAJN
removed /dnsaddr/bootstrap.libp2p.io/p2p/QmQCU2EcMqAqPR2i9bChD9tGN3chTbq5TbXJ16u19uLTa
removed /dnsaddr/bootstrap.libp2p.io/p2p/QmbLHAnMoJPWSCR5Zhtx6BHX9KiKWN6tpvUqcqanj75Nb
removed /dnsaddr/bootstrap.libp2p.io/p2p/QmcZf59bWkK5XFi76CZ8cbJ4BhTzzA3gU1ZjYzCYM3dwt
removed /ip4/104.131.131.82/tcp/4001/p2p/QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbUtfsmvsgQLuuvJ
removed /ip4/104.131.131.82/udp/4001/quiic-v1/p2p/QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbUtfsmvsgQLuuvJ
Cloning into 'go-ipfs-swarm-key-gen'...
Initializing daemon...
Kubo version: 0.28.0
Repo version: 15
System version: amd64/linux
Golang version: go1.22.5
2024-07-18T09:23:21.595Z ERROR cmd/ipfs kubo/daemon.go:40Private networking (swarm.key / LIBP2P_FORCE_PNET) does not
use Routing.Type=dht instead. Update config to remove this message.
Swarm is limited to private network of peers with the swarm key
Swarm key fingerprint: 0b8c1ad3cfee8521f171cc259ddef9c
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/172.17.0.2/tcp/4001
Swarm listening on /p2p-circuit
Swarm announcing /ip4/127.0.0.1/tcp/4001
Swarm announcing /ip4/172.17.0.2/tcp/4001
RPC API server listening on /ip4/127.0.0.1/tcp/5001
WebUI: http://127.0.0.1:5001/webui
Gateway server listening on /ip4/127.0.0.1/tcp/8080
Daemon is ready

```

Figura B.8.: Logs de la ejecución del nodo del oráculo.

B.1.3 Oráculo

En primer lugar, es necesario compilar el contrato `oraculo.sol` y desplegarlo en la red de Hardhat utilizando Remix IDE o cual otra herramienta que permita compilar y desplegar smart contracts. Una vez hecho esto hay que copiar el ABI y el address y pegarlos en `oraculoNodo.py`.

Dentro del Dockerfile se definen los procesos necesarios para preparar el contenedor para la ejecución del oráculo y del nodo de IPFS. Dentro del `script.sh` se definen las operaciones que realiza el contenedor una vez desplegado, como la generación de la `swarm.key`, la ejecución del oráculo y el nodo IPFS.

Para crear la imagen del contenedor se utiliza el comando:

```
1 docker build -t nombre_imagen .
```

Para desplegar el contenedor e iniciar el oráculo se utiliza el comando:

```
1 docker run --name nombre_contenedor -p 8000:8000 nombre_imagen
```

Una vez desplegado el contenedor, se nos mostrará por consola los datos de la inicialización del nodo IPFS, como se puede ver en la Fig. B.8.

El proceso comienza generando un par de claves ED25519 para la autenticación y encriptación, seguido de la identificación del peer en la red IPFS. Se inicializa un

nodo IPFS en el directorio `"/root/.ipfs"`. Posteriormente, se eliminan varias direcciones de bootstrap predeterminadas que se utilizan para descubrir otros peers en la red IPFS. Se clona el repositorio `'go-ipfs-swarm-key-gen'`, para generar una clave de swarm para la red privada. Luego, se inicia el proceso daemon de IPFS, mostrando las versiones utilizadas. La red de peers (swarm) está limitada a una red privada utilizando la clave de swarm generada. El nodo está escuchando conexiones en las direcciones IP 127.0.0.1 y 172.17.0.2 en el puerto TCP 4001, así como utilizando el protocolo `'p2p-circuit'`, anunciando su disponibilidad en las mismas direcciones y puerto.

B.1.4 Front-End

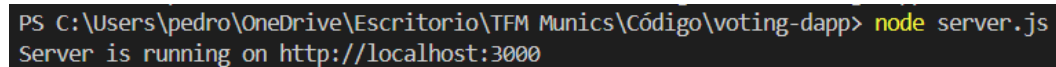
Para levantar el front-end, en primer lugar es necesario asegurarse de que Node.js y npm se encuentran instalados en el equipo. Para instalar las dependencias del proyecto, se puede ejecutar el siguiente comando en el directorio raíz (voting-dapp):

```
1 npm install
```

Para levantar el front-end, es suficiente con ejecutar el siguiente comando:

```
1 node server.js
```

Esto levantará la aplicación web en el puerto 3000, como se puede ver en la Fig. B.9



```
PS C:\Users\pedro\OneDrive\Escritorio\TFM Munics\Código\voting-dapp> node server.js
Server is running on http://localhost:3000
```

Figura B.9.: Logs de la ejecución del front-end.

B.2 Walkthrough

Una vez desplegado cada uno de los componentes que conforman el sistema, podremos acceder a la aplicación web a través de la dirección `localhost:3000`. Una vez dentro, con la cuenta que está autorizada para ejercer de administrador (en este caso, la cuenta número 19 de las generadas por Hardhat), se nos mostrarán las opciones para desplegar un contrato de votación, como se puede ver en la Fig. B.10.

Deploy Voting DApp

Title:

Option 1 Name:

Option 2 Name:

Figura B.10.: Vista de la aplicación web de los elementos para crear una votación.

Una vez completados, a través de la aplicación web, los diferentes parámetros del contrato, MetaMask nos mostrará una vista previa de la transacción para que podamos confirmarla o rechazarla, mostrada en la Fig. B.11. Una vez confirmada, se desplegará el contrato, se realizará una llamada al nodo de OrbitDB y se almacenará la dirección en la que se encuentra el contrato desplegado, como se muestra en la Fig. B.12.

The screenshot shows the MetaMask mobile interface for a transaction confirmation. At the top, it displays the account 'Hardhat 8544' and the role 'Administrador'. Below this, the URL 'http://localhost:3000' is shown, followed by a button labeled 'IMPLEMENTACIÓN DE CONTRATO'. The transaction details are presented in a dark-themed box with two main sections: 'Estimated fee' and 'Total'. Both sections show a value of '0.045 HARDHATETHER' and a maximum value of '0.045 HARDHATETHER'. At the bottom, there are two buttons: 'Rechazar' (Reject) and 'Confirmar' (Confirm).

Category	Value	Maximum Value
Estimated fee	0.045 HARDHATETHER	0.045 HARDHATETHER
Total	0.045 HARDHATETHER	0.045 HARDHATETHER

Figura B.11.: Vista de la transacción de MetaMask.

```
Nuevo registro añadido: { key: "0xC5888275e0a1ca13a26463318105957aa4d1feD7", value: "Bipartito", hash: "zdpuArRikiffXrCg5d3YRD6mLHMukMjrt57gsm9YME4PKSWH1" }
Nuevo registro añadido: { key: "0x2d1309Fde5D8c1ab2c8036c26FadfE9d933Ce9E4", value: "Balon de oro", hash: "zdpuAyLP1KMx8SfuUbycDxpn3qRSgG3zQ5wclWEArKfcORDZRw" }
```

Figura B.12.: Captura de los registros de la base de datos de OrbitDB, tal y como se muestran en tiempo real en uno de los nodos.

Al recargar la página, esta vez desde una cuenta diferente que tenga el rol de votante, podremos ver la vista de la aplicación web con los elementos que nos permiten elegir las votaciones abiertas, como se muestra en la Fig. B.13. Al cargar la página, se realiza una llamada al nodo de OrbitDB, que se encargará de devolver la lista de todos los contratos que se encuentran desplegados en ese momento. Al seleccionar una de las opciones de los contratos en el desplegable, se realizará una llamada al contrato asociado y se recuperarán los diferentes parámetros que contiene este smart contract. Haciendo uso de los tres botones que se nos muestran, podemos votar por alguna de las opciones o ver los resultados.

Deployed Contracts

0x73511669fd4dE447feD18BB79bAFeAC93aB7F31f: Bipartito ▾

Title:

Option 1:

Option 2:

Figura B.13.: Vista de la aplicación web de los elementos para elegir votaciones abiertas y votar, tal y como se ve desde la vista del votante.

Si volvemos a la página esta vez utilizando MetaMask con la cuenta de administrador que usamos inicialmente, se nos mostrarán las opciones para ver los resultados o cerrar la votación, como se muestra en la Fig. B.14. Al igual que a la hora de desplegarla, MetaMask nos mostrará una ventana que nos permite confirmar o rechazar la transacción, mostrada en la Fig. B.15. Una vez cerrada la votación, únicamente podremos ver los resultados de la votación, pero no podremos votar por ninguna de las opciones. Tanto desde una cuenta de administrador como desde una cuenta de votante, la vista será la mostrada en la Fig. B.16. Si queremos ver los resultados de la votación, serán mostrados a través de una alerta como se muestra en la Fig. B.17.

Deployed Contracts

0xC469e7aE4aD962c30c7111dc580B4adbc7E914DD: Bipartito ▾

Title:

Option 1:

Option 2:

Figura B.14.: Vista de la aplicación web de los elementos para elegir votaciones abiertas y cerrarlas, tal y como se ve desde la vista del administrador.

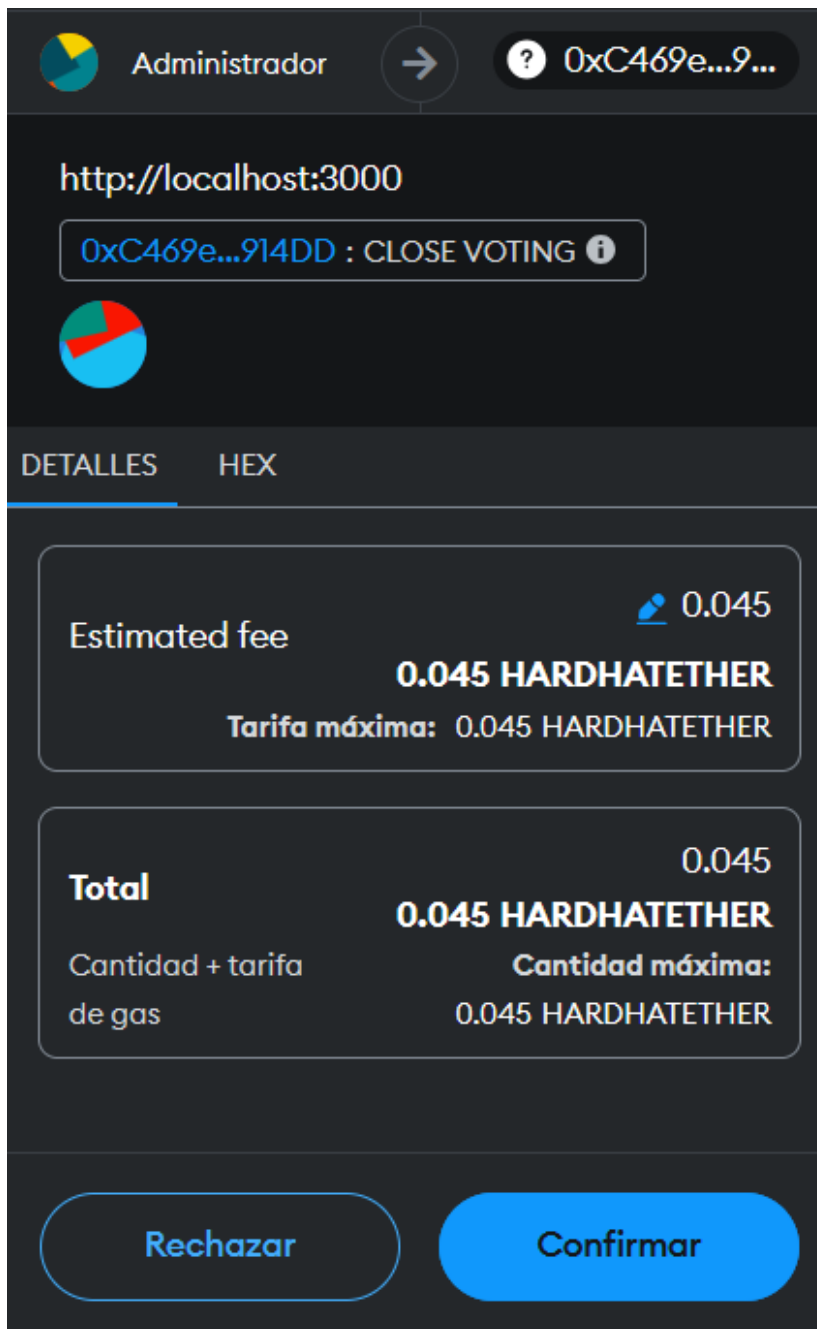


Figura B.15.: Vista de MetaMask que permite aceptar o rechazar la transacción de cerrar el contrato.

Deployed Contracts

0xC469e7aE4aD962c30c7111dc580B4adbc7E914DD: Bipartito ▾

Title:

Option 1:

Option 2:

Figura B.16.: Vista de una votación que ya ha sido cerrada.

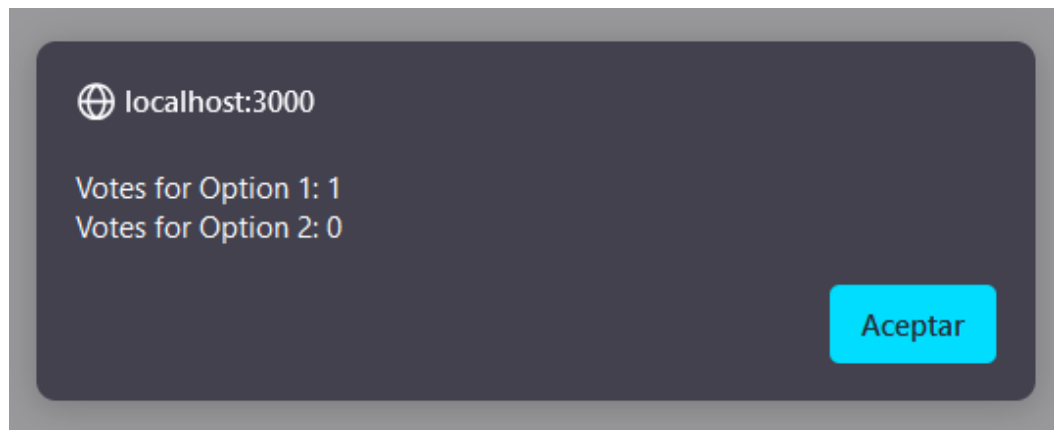


Figura B.17.: Vista de la alerta que muestra el recuento de votos.

Bibliografía

- [24a] *Agora: Bringing voting systems into the digital age*. 2024. URL: <https://www.agora.vote/> (visitado 3 de jul. de 2024) (vid. pág. 9).
- [Ben20] Abdeljalil Beniiche. *A Study of Blockchain Oracles*. 2020. URL: <https://arxiv.org/pdf/2004.07140> (vid. pág. 64).
- [24b] *BLOCKO*. 2024. URL: <https://en.blocko.io/> (visitado 3 de jul. de 2024) (vid. pág. 9).
- [EGJ18] Parinya Ekparinya, Vincent Gramoli y Guillaume Jourjon. *Impact of Man-In-The-Middle Attacks on Ethereum*. 2018. URL: <https://ieeexplore.ieee.org/document/8613949> (vid. pág. 68).
- [24c] *eth-phishing-detect*. 2024. URL: <https://github.com/MetaMask/eth-phishing-detect> (visitado 6 de mayo de 2024) (vid. pág. 48).
- [24d] *Ethereum*. 2024. URL: <https://ethereum.org/es/> (visitado 3 de jul. de 2024) (vid. págs. 10, 12).
- [24e] *Ethereum official documentation on oracles*. 2024. URL: <https://ethereum.org/en/developers/docs/oracles> (visitado 7 de mar. de 2024) (vid. pág. 34).
- [FS03] Niels Ferguson y Bruce Schneier. *Practical Cryptography*. John Wiley y Sons, 2003 (vid. pág. 34).
- [FF18] Tiago M. Fernández-Caramés y Paula Fraga-Lamas. “A Review on the Use of Blockchain for the Internet of Things”. En: *IEEE Access* (2018) (vid. pág. 10).
- [FF20] Tiago M. Fernández-Caramés y Paula Fraga-Lamas. “Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks”. En: *IEEE Access* (2020) (vid. pág. 70).
- [Fou07] Free Software Foundation. *GNU GENERAL PUBLIC LICENSE v3*. 2007. URL: <https://www.gnu.org/licenses/gpl-3.0.html> (vid. pág. 81).
- [Fou15] Linux Foundation. *Hyperledger Fabric*. 2015. URL: <https://www.hyperledger.org/projects/fabric> (visitado 3 de jul. de 2024) (vid. pág. 11).
- [Gar+24] Pedro García Cereijo, Gabriel Fernández-Blanco, Paula Fraga-Lamas y Tiago M. Fernández-Caramés. “EtherTuna: A Pseudo-Random Number Generator Oracle for Ethereum Networks Based on Fortuna (aceptado en BCCA 2024)”. En: *2024 6th International Conference on Blockchain Computing and Applications (BCCA)* (2024) (vid. págs. 34, 36).

- [Gar+18] Alberto Garoffolo, Pier Stabilini, Robert Viglione y Uri Stav. *A Penalty System for Delayed Block Submission*. 2018. URL: <https://www.horizen.io/assets/files/A-Penalty-System-for-Delayed-Block-Submission-by-Horizen.pdf> (vid. pág. 67).
- [Gro96] Lov K. Grover. *A fast quantum mechanical algorithm for database search*. 1996 (vid. pág. 69).
- [18a] *Hackers emptied Ethereum wallets by breaking the basic infrastructure of the internet*. 2018. URL: <https://www.theverge.com/2018/4/24/17275982/myetherwallet-hack-bgp-dns-hijacking-stolen-ethereum> (vid. pág. 68).
- [24f] *Hardhat*. 2024. URL: <https://hardhat.org/> (visitado 3 de jul. de 2024) (vid. pág. 14).
- [24g] *Helia*. 2024. URL: <https://github.com/ipfs/helia> (visitado 3 de jul. de 2024) (vid. pág. 44).
- [Hja+18] Friorik Hjalmarrsson, Gunnlaugur Hreioarsson, Mohammad Hamdaq y Gisli Hjálmtýsson. “Blockchain-Based E-Voting System”. En: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (2018) (vid. pág. 75).
- [24h] *IPFS*. 2024. URL: <https://ipfs.tech/> (visitado 3 de jul. de 2024) (vid. pág. 15).
- [JAS21] Uzma Jafar, Mohd Juzaidin Ab Aziz y Zarina Shukur. *Blockchain for Electronic Voting System Review and Open Research Challenges*. 2021. URL: <https://www.mdpi.com/1424-8220/21/17/5874> (vid. pág. 8).
- [18b] *Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales (LOPDGDD)*. 2018. URL: <https://www.boe.es/eli/es/lo/2018/12/05/3/con> (visitado 3 de jul. de 2024) (vid. pág. 20).
- [85] *Ley Orgánica del Régimen Electoral General (LOREG)*. 1985. URL: <https://www.boe.es/eli/es/lo/1985/06/19/5/con> (visitado 3 de jul. de 2024) (vid. pág. 20).
- [17] *Local Government in South Korea Taps Blockchain for Community Vote*. 2017. URL: <https://www.coindesk.com/markets/2017/03/07/local-government-in-south-korea-taps-blockchain-for-community-vote/> (visitado 3 de jul. de 2024) (vid. pág. 9).
- [24i] *Metamask*. 2024. URL: <https://metamask.io/> (visitado 3 de jul. de 2024) (vid. pág. 14).
- [Nak08] Satoshi Nakamoto. *Bitcoin*. 2008. URL: <https://bitcoin.org/es/> (vid. pág. 10).
- [Nak09] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. En: *Cryptography Mailing list at https://metzdowd.com* (2009) (vid. pág. 10).
- [24j] *Node.js*. 2024. URL: <https://nodejs.org/en> (visitado 3 de jul. de 2024) (vid. pág. 13).
- [24k] *OpenZeppelin Contracts*. 2024. URL: <https://www.openzeppelin.com/contracts> (visitado 2 de feb. de 2024) (vid. pág. 30).

- [24l] OrbitDB. 2024. URL: <https://orbitdb.org/> (visitado 3 de jul. de 2024) (vid. pág. 15).
- [R317] R3. Corda. 2017. URL: <https://corda.net/> (visitado 3 de jul. de 2024) (vid. pág. 11).
- [16] *Reglamento General de Protección de Datos (RGPD)*. 2016. URL: <https://www.boe.es/doue/2016/119/L00001-00088.pdf> (visitado 2 de feb. de 2024) (vid. pág. 20).
- [24m] Remix IDE. 2024. URL: <https://remix-project.org> (visitado 7 de jul. de 2024) (vid. pág. 55).
- [24n] Repositorio GitHub: Sistema de votación electrónica basado en blockchain. 2024. URL: https://github.com/pedrogarciacereijo/TFM_Munics (vid. pág. 81).
- [Rho81] Peter J Rhodes. "Notes on voting in Athens". En: *Greek, Roman, and Byzantine Studies* (1981) (vid. pág. 7).
- [Rip12] Inc. Ripple Labs. *Ripple*. 2012. URL: <https://ripple.com> (visitado 3 de jul. de 2024) (vid. pág. 11).
- [SM19] Sarwar Sayeed y Hector Marco-Gisbert. *Assessing Blockchain Consensus and Security Mechanisms against the 51 Attack*. 2019. URL: <https://doi.org/10.3390/app9091788> (visitado 2 de feb. de 2024) (vid. pág. 66).
- [Sho97] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". En: *SIAM Journal on Computing* (1997) (vid. pág. 69).
- [18c] *Sierra Leone just became the first country in the world to use blockchain during an election*. 2018. URL: <https://www.businessinsider.com/sierra-leone-blockchain-elections-2018-3> (visitado 3 de jul. de 2024) (vid. pág. 9).
- [18d] *Sierra Leone's blockchain vote sounds neat, but dont get carried away*. 2018. URL: <https://www.technologyreview.com/2018/03/13/144709/sierra-leones-blockchain-vote-sounds-neat-but-dont-get-carried-away/> (visitado 3 de jul. de 2024) (vid. pág. 9).
- [24o] Solidity. 2024. URL: <https://soliditylang.org/> (visitado 3 de jul. de 2024) (vid. pág. 13).
- [24p] *Voto electrónico en el mundo*. 2024. URL: <https://www.euskadi.eus/informacion/voto-electronico-voto-electronico-en-el-mundo/web01-a2haukon/es/> (visitado 2 de feb. de 2024) (vid. pág. 8).
- [ZNP15] Guy Zyskind, Oz Nathan y Alex 'Sandy' Pentland. *Decentralizing Privacy: Using Blockchain to Protect Personal Data*. 2015 (vid. pág. 9).

