Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Diseño e implementación de un sistema de votación en línea basado en blockchain con cifrado homomórfico

Design and implementation of an online voting system based on blockchain and homomorphic encryption

Javier Padilla Pío

La Laguna, 24 de mayo de 2024

D. **Cándido Caballero Gil**, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Jezabel Miriam Molina Gil**, profesora Contratado Doctor de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Diseño e implementación de un sistema de votación en línea basado en blockchain con cifrado homomórfico*”

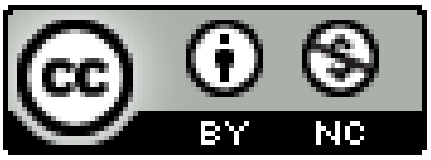
ha sido realizada bajo su dirección por D. **Javier Padilla Pío**

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 24 de mayo de 2024

Agradecimientos

A mi familia y amigos, que han mostrado un apoyo incondicional en todo lo que hago.

Licencia



Resumen

El objetivo principal de este trabajo es desarrollar un protocolo de votación electrónico aprovechando las características del Blockchain y la criptografía homomórfica.

Todo sistema de votación debe garantizar el anonimato del votante junto con la integridad del sistema, de tal manera que nadie pueda saber el voto de otra persona, pero se garantice que su voto ha sido contabilizado correctamente. Además, se debe garantizar que no se conoce el resultado de la misma hasta el momento del escrutinio, puesto que podría modificar el resultado.

Para poder garantizar estas condiciones, todos los votos serán cifrados en el cliente y el servidor no podrá relacionarlo con el usuario, ya sea cifrado o sin cifrar. Además, asemejando los sistemas tradicionales, serán los usuarios los que depositen su voto cifrado en la "urna", que será la cadena de bloques. De esta manera todo el almacenamiento y posterior procesamiento de los votos para el recuento será inmutable y transparente de cara al usuario.

Este protocolo propuesto se ha implementando creando una plataforma denominada "E3VOTE", en la que se podrán crear y administrar votaciones, dando permisos a los usuarios para votar y manejando todo el cifrado y verificación de los votos.

Internamente se ha desplegado también una cadena de bloques privada con ciertas modificaciones respecto de la red principal de Ethereum para adecuarse a las necesidades de la aplicación.

Palabras clave: contrato, votación, elección, NextJS, React, PostgreSQL, Ethereum, Blockchain, Solidity, ElGamal, RSA, TypeScript, Rust, Go.

Abstract

The main goal of this work is to develop an electronic voting protocol taking advantage of Blockchain and homomorphic cryptography features.

Any voting system must guarantee the anonymity of the voter along with the integrity of the system, so that no one can know the vote of another person, but it is also guaranteed that his vote has been tallyed correctly. In addition, it must be guaranteed that the result of the vote is not known until the time of counting, since it could change the result.

In order to guarantee these conditions, all votes will be encrypted on the client and the server will not be able to relate it to the user, whether encrypted or unencrypted. Furthermore, resembling traditional systems, it will be the users who will deposit their encrypted vote in the "ballot box" which will be the blockchain. In this way all the storage and subsequent processing of the votes for counting will be immutable and transparent to the user.

This proposed protocol has been implemented by creating a platform called "E3VOTE", where elections can be created and managed, giving permissions to users to vote and handling all the encryption and verification of votes.

Internally, a private blockchain has also been deployed with certain modifications to the Ethereum mainnet to suit the needs of the application.

Keywords: contract, ellection, NextJS, React, PostgreSQL, Ethereum, Blockchain, Solidity, ElGamal, RSA, TypeScript, Rust, Go.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En la actualidad, los procesos electorales desempeñan un papel crucial en la configuración de gobiernos y organizaciones democráticas. Tradicionalmente, se han llevado a cabo de manera presencial, utilizando papeletas y urnas físicas. Este método, aunque efectivo, no está exento de desafíos. Estos protocolos requieren una infraestructura considerable, incluyendo la preparación y supervisión de los centros de votación, la impresión y distribución de papeletas y la posterior recolección y recuento de votos. Estos pasos son, además de costosos, susceptibles a errores humanos y fraudes.

Además, los métodos tradicionales pueden dificultar la participación de ciertos grupos de la población. Las personas con movilidad reducida, aquellos que viven en áreas remotas o los ciudadanos que se encuentran fuera del país durante las elecciones pueden enfrentarse a barreras significativas para ejercer su derecho al voto. En consecuencia, la búsqueda de métodos alternativos, más accesibles y eficientes se ha convertido en una prioridad.

En este contexto, los sistemas de votación electrónica emergen como una solución prometedora. Pueden simplificar y agilizar el proceso electoral, reduciendo costes y minimizando errores. Un sistema de votación online podría mejorar la accesibilidad, permitiendo que un mayor número de ciudadanos participe en las elecciones, con mayor comodidad y sin las restricciones ya vistas.

1.1. Problemas que plantea un sistema electrónico

A pesar de los numerosos beneficios que los sistemas de votación electrónica pueden ofrecer, su implementación también presenta una serie de inconvenientes que deben abordarse para garantizar su eficacia y seguridad. Entre los problemas más destacados se encuentran los siguientes:

- **Seguridad y vulnerabilidades.** Introducir una dependencia en la tecnología hace que el sistema pueda ser atacado ya sea para obtener información o simplemente de manera disruptiva.
- **Privacidad y Confidencialidad.** Se debe garantizar a toda costa la privacidad de un usuario. Nadie debe saber su voto, ni tan siquiera la administración de la

votación.

- **Transparencia.** Todo el proceso de votación se debe hacer de manera transparente al usuario, pudiendo ser verificado por él o una entidad externa de confianza.
- **Accesibilidad.** Si bien permitir que los usuarios voten telemáticamente facilitaría el proceso a una parte de la población, esto podría excluir a un grupo que no cuente con la tecnología o conocimientos necesarios para poder ejercer su derecho al voto.

Vista esta problemática, por mucho que se consiguieran solventar todos los puntos mencionados, quedaría obtener la confianza de la sociedad. Esto no es algo que se pueda definir en el protocolo, sino que viene derivado de solucionar todos los posibles problemas que podría tener un nuevo sistema electrónico.

1.1.1. Estado del Arte

Actualmente se han implementado distintos protocolos que afrontan esta problemática. Algunos ejemplos pueden ser Venezuela, Argentina o Estados Unidos, que cuentan con máquinas para emitir el voto de manera electrónica en los Colegios Electorales. No obstante, a día de hoy, solo Estonia cuenta con un sistema de votación a través de internet el cual se ha utilizado para votaciones a nivel parlamentario.

El protocolo ha sido desarrollado por la empresa Smartmatic (1), pionera en el desarrollo de automatizaciones para todo el proceso electoral. Implanta soluciones a distintos niveles en países de todo el mundo, desde América del Sur hasta Oceanía. No obstante la mayoría de estas soluciones, a excepción de Estonia, no implementan el voto telemático sino automatizaciones, como máquinas de recuento o máquinas para votar electrónicamente de manera presencial.

El sistema estonio se utilizó por primera vez en 2005 con un $1{,}9\,\%$ de participación que ha ido creciendo hasta un $51\,\%$ en 2023. La definición del protocolo es de acceso público y el código fuente del servidor se hizo público en 2013 debido a la presión social por parte de la comunidad científica. No obstante, el código para el cliente no ha sido publicado puesto que se considera que abriría la puerta a posibles ataques.

Los votantes pueden verificar que su voto se ha emitido correctamente en el momento de votar, pero no pueden verificar que forme parte del recuento final. Aunque se han implementado diversas medidas para garantizar la transparencia del sistema, no se puede garantizar que el recuento y el almacenamiento no sea alterado en los propios servidores.

La verificación del voto se puede realizar, puesto que al cifrar la papeleta se utiliza un algoritmo determinista al que se le añade un relleno aleatorio para después poder reconstruirla. A este proceso se le conoce como prueba de conocimiento del texto original. Internamente, los votos se guardarán cifrados en un servidor y en el momento del recuento se exportarán a un DVD. Este disco se introducirá en el servidor de recuento, que utilizando un HSM (2) que contiene la clave privada de la votación, descifrará cada voto y después los sumará. El recuento podría suponer una grave violación de la privacidad del usuario, puesto que en ese servidor se podrá asociar el candidato elegido con el votante.

En cuanto a estudios académicos, el trabajo (3) examina en profundidad los desafíos que presenta el voto electrónico basado en blockchain, incluyendo la privacidad, la escalabilidad, la seguridad y la interoperabilidad. Los autores discuten las soluciones potenciales y las áreas que requieren mayor investigación.

Los autores de (4) realizan una revisión sistemática que analiza las diferentes propuestas de voto electrónico basadas en blockchain, evaluando sus características, ventajas y desventajas. Los autores identifican los retos y oportunidades clave para la adopción de esta tecnología en el ámbito electoral. Por otro lado, (5) hace una revisión bibliográfica para explorar las aplicaciones potenciales de blockchain en el voto electrónico, enfocándose en la mejora de la transparencia, la seguridad y la eficiencia. Los autores discuten los desafíos técnicos y legales que deben abordarse para implementar estas soluciones.

En la tesis (6) se analiza en detalle la seguridad de una propuesta específica de voto electrónico con blockchain. La autora evalúa la resistencia a diversos ataques y propone mejoras para fortalecer la seguridad del sistema.

Los trabajos (7), (8) y (9) realizan diferentes propuestas. En estas propuestas los autores evalúan el rendimiento y la seguridad de los prototipos, intentan garantizar la privacidad del voto y la integridad del proceso electoral. Implementan contratos inteligentes para gestionar el proceso de votación, y mencionan la ventajas de emitir su voto desde cualquier dispositivo con acceso a internet, así como sus posibles problemas e implicaciones.

En cuanto a sistemas de votación electrónica usando criptografía homomórfica, los autores de (10), mencionan como la criptografía homomórfica permite contar los votos sin revelar su contenido individual. Esto mejora la privacidad del voto y la seguridad del proceso electoral. El trabajo (11) también hace uso de pruebas de conocimiento cero para verificar la identidad del votante sin revelar su voto. Por su parte, los autores de (12), explican su esquema en el que se permite votar de forma segura y auditable a través de internet. El sistema utiliza técnicas de agregación de firmas para garantizar la integridad del proceso electoral.

En este trabajo se busca implementar la tecnología Blockchain (13) con el objetivo de ofrecer el grado de transparencia necesario para todo protocolo de votación, el cual no se ha alcanzado en un sistema real aún. Además, se aprovecharán las características de los cifrados homomórficos, para que en ningún momento sea necesario descifrar un voto y poder revelar así a quién ha votado un usuario.

Capítulo 2

Tecnologías utilizadas

En este capítulo se presentan una pequeña introducción a las distintas tecnologías sobre las que se sustenta este trabajo.

2.1. Criptografía asimétrica

Dentro de la criptografía podemos distinguir dos grandes tipos de criptosistema: simétricos y asimétricos. Los simétricos son aquellos que, usando la confusión y difusión, son capaces de cifrar un mensaje de tal manera que la misma clave utilizada para cifrar se use para descifrar. Por otro lado, los criptosistemas asimétricos utilizan el álgebra modular y la teoría de números para que la clave de cifrado y descifrado no sean la misma.

Todos los criptosistemas asimétricos se basan en la misma idea: una función unidireccional con trampa. Esta función es muy fácil de calcular en un sentido pero, en el opuesto es computacionalmente inabordable, a no ser que se conozca un valor secreto conocido como trampa. Algunos ejemplos pueden ser la factorización de números enteros, el logaritmo discreto, etc.

2.1.1. RSA

Este sistema basado en la factorización de números enteros es hoy en día uno de los más usados para firmar digitalmente.

1. Se eligen dos números primos distintos p y q.
2. Se calcula el producto de los números primos (n) que servirá como módulo para todas las futuras operaciones.
3. Utilizando la función de Euler φ se calcula $\varphi(n) = (p-1)(q-1)$. A continuación, se escoge un entero positivo (e) menor que $\varphi(n)$ y coprimo con $\varphi(n)$.
4. Utilizando aritmética modular (14) se escoge un número d tal que $e \cdot d \equiv 1 \pmod{\varphi(n)}$.

Finalmente para cifrar y descifrar se usarán los exponentes e y d, operando siempre en módulo n.

$$c = m^e \ (mod\, n)$$
$$m = c^d \ (mod\, n)$$

Firma ciega

Un mecanismo de firma ciega permite a un usuario firmar un mensaje que ha sido cegado, de tal manera que solo el remitente puede deshacer esta acción para quedarse con el mensaje firmado por el servidor.

Una analogía para entender este concepto podría ser el caso de un usuario Alice que quiere que Bob firme un documento. Lo guarda en un sobre hecho de papel de calco y se lo envía a Bob; este firmará el sobre y esa firma se calcará al documento. De esta manera Bob nunca llegará a ver el documento firmado.

Aprovechando las propiedades de las potencias y la relación que existe entre e y d, se puede desarrollar un sistema en el que el propietario de la clave privada firme un mensaje cegado.

$$m' = mr^e \ (mod\, n)$$
$$s' = m'^d \ (mod\, n)$$
$$s = s'r^{-1} \ (mod\, n) = m^d r^{ed} r^{-1} \ (mod\, n) = m^d r r^{-1} \ (mod\, n) = m^d \ (mod\, n)$$

2.1.2. ElGamal

Este criptosistema se basa en otro problema computacionalmente inabordable: el logaritmo discreto. Este problema ha sido estudiado en profundidad y se ha podido extrapolar a las curvas elípticas. Con esto se consigue un nivel de seguridad similar con tamaños de clave mucho más pequeños.

1. Alice elige un exponente privado a y un número público g. Con estas dos componentes genera la clave pública. $K = g^a$
2. Bob encripta el mensaje m junto con un número aleatorio r.
$$E(m) = (K^r \cdot m,\ g^r)$$
3. Alice utiliza la clave privada para eliminar la componente aleatoria del mensaje,
$$m = g^{-a} \cdot g^r \cdot K^r \cdot m = g^{-ar} \cdot g^{ar} \cdot m = m$$

El uso del número aleatorio no solo ofrece un grado extra de seguridad puesto que hace que el criptosistema no sea determinista (al contrario que RSA), sino que abre la puerta a que el usuario puede verificar que ha generado un texto cifrado sabiendo la componente aleatoria utilizada.

2.1.3. Cifrados homomórficos

Se dice que un sistema es homomórfico si cumple que $E(f(a,b)) = f(E(a), E(b))$, siendo E la función de cifrado y f una función cualquiera. Si desarrollamos la función de cifrado de ElGamal podemos llegar a la conclusión que es homomórfico para la multiplicación:

$$E(m_1) \cdot E(m_2) = (K^{r_1} \cdot m_1,\ g^{r_1}) \cdot (K^{r_2} \cdot m_2,\ g^{r_2}) =$$

$$= ((m_1 \cdot m_2)K^{r_1+r_2},\ g^{r_1+r_2}) = E(m_1 \cdot m_2)$$

Esta propiedad podemos fácilmente modificarla para obtener una variante de ElGamal homomórfico para la suma, basta con exponenciar el mensaje a g.

$$E(g^{m_1}) \cdot E(g^{m_2}) = (K^{r_1} \cdot g^{m_1},\ g^{r_1}) \cdot (K^{r_2} \cdot g^{m_2},\ g^{r_2}) =$$
$$= (g^{(m_1+m_2)}K^{r_1+r_2},\ g^{r_1+r_2}) = E(g^{m_1+m_2})$$

En este trabajo se usará una versión de ElGamal exponencial sobre curva elíptica para optimizar los cálculos. A continuación se especifica la nueva notación, minúsculas para escalares y mayúsculas para puntos en la curva.

$$E(m) = (rG, mG + rK)$$

G es el punto generador que será público y K la clave pública generada multiplicando un escalar privado a por el generador. Con este nuevo sistema se puede demostrar también la propiedad homomórfica en la suma.

$$E(m_1) + E(m_2) = (r_1G,\ m_1G + r_1K) + (r_2G,\ m_2G + r_2K) =$$

$$= ((r_1 + r_2)G,\ (m_1 + m_2)G + (r_1 + r_2)K) = E(m_1 + m_2)$$

2.1.4. ZKP: Pruebas de conocimiento nulo

En criptografía, una prueba de conocimiento nulo (Zero-Knowledge Proof, ZKP) es un método por el cual una parte (el probador) demuestra a otra parte (el verificador) que posee cierto conocimiento, sin revelar ninguna información sobre dicho conocimiento.

Características

- Completitud: Si la declaración es verdadera, un probador honesto puede convencer al verificador honesto de esta verdad.
- Solvencia: Si la declaración es falsa, ningún probador deshonesto puede convencer al verificador honesto de que es verdadera, excepto con una probabilidad muy pequeña.
- Conocimiento Nulo: Si la declaración es verdadera, el verificador no aprende nada más allá de la veracidad de la declaración. Es decir, el verificador no obtiene ninguna información adicional sobre el conocimiento en sí mismo.

2.2. NextJS

Tanto la interfaz web como el servidor encargado de interactuar con la cadena de bloques y autenticar a los usuarios ha sido desarrollada usando NextJS. Este framework de TypeScript (15) basado en React (16) cuenta con una serie de características que lo hacen ideal para cualquier proyecto que implique una interfaz web.

- Renderizado Híbrido:
  - Server-side Rendering (SSR): Genera HTML (17) en el servidor por cada solicitud. Esto que mejora el SEO (18) y reduce el tiempo de carga inicial.
  - Static Site Generation (SSG): Genera HTML en el momento de la compilación, lo que permite las páginas de manera estática.
  - Client-side Rendering (CSR): Similar a una aplicación típica de React, donde el HTML se genera en el navegador del usuario.
- Incremental Static Regeneration (ISR): Permite actualizar páginas estáticas después de la compilación inicial, proporcionando flexibilidad para manejar datos dinámicos y estáticos juntos.
- Rutas Automáticas: El sistema de enrutamiento de NextJS se basa en el sistema de archivos en el que dependiendo del nombre del archivo este tendrá una finalidad concreta: interfaces compartidas, enrutamiento anidado, pantallas de carga o error, etc. La ruta de la página se generará a partir de la ruta hasta el archivo que contiene su código que por convención debe llamarse “page.ts” o “page.js”.

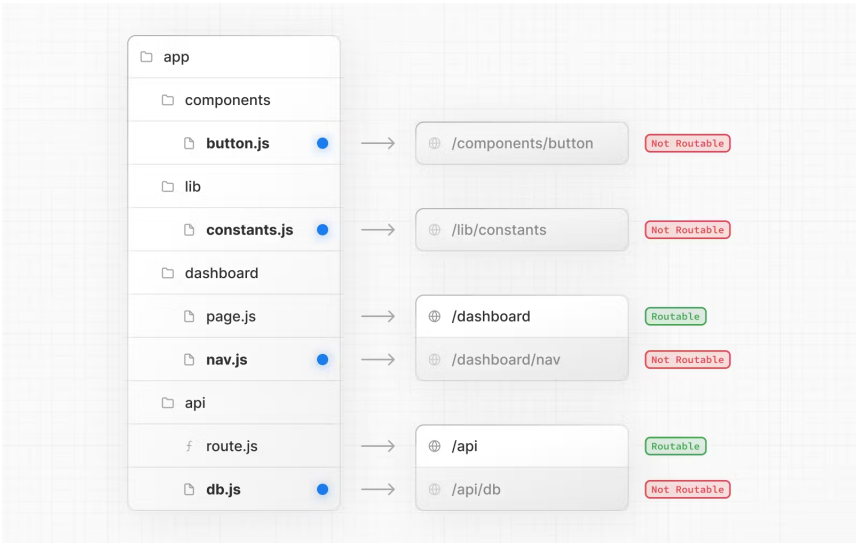


Figura 2.1: Ejemplo rutas nextjs

- Soporte para API Integradas: Permite crear rutas API dentro de la misma aplicación, proporcionando una solución completa para desarrollar tanto el frontend como el backend.
- Server Actions: Abstrae la conexión HTTP (19) que se crearía al realizar una petición desde el cliente a una API. Actúan como funciones asíncronas ejecutadas desde el cliente.

- Partial rendering: NextJS permite que al navegar entre rutas parte de la interfaz se conserve. Como se puede ver en la figura 2.2 al navegar a invoices solo cambiará el interior del contenedor ubicado a la derecha. Esto permite preservar estado en el cliente entre distintas rutas, siempre y cuando se almacene en la parte de la interfaz compartida.

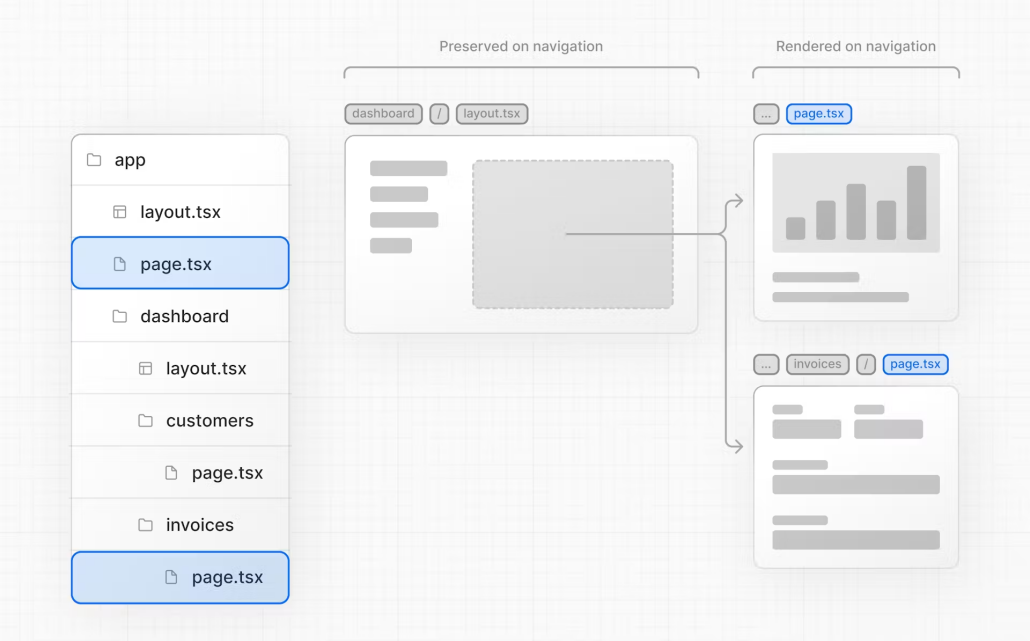


Figura 2.2: Partial rendering

2.3. Ethereum

Ethereum es una plataforma basada en Blockchain ideada para la creación de aplicaciones descentralizadas (dApps). Propuesta por Vitalik Buterin en 2013 (20) y lanzada en 2015, Ethereum se diferencia de otras Blockchains como Bitcoin (21) por su capacidad de soportar un amplio rango de aplicaciones más allá de simples transacciones financieras. Internamente, Ethereum utiliza su propia moneda, Ether (ETH), que se emplea para pagar las tarifas de transacción y los servicios de computación en la red.

Una de las principales funcionalidades de Ethereum son los contratos inteligentes, que son programas autoejecutables donde los términos del acuerdo se codifican directamente. Esto permite que las transacciones y acuerdos se realicen de manera automática cuando se cumplan las condiciones predefinidas, eliminando la dependencia en entidades de confianza. Gracias a esto, Ethereum se utiliza para una amplia gama de aplicaciones, desde finanzas descentralizadas (DeFi), donde se pueden realizar préstamos e intercambios sin bancos, hasta tokens no fungibles (NFTs), que representan la propiedad de activos digitales únicos como arte y música.

Los usuarios interactúan con la cadena utilizando un par de claves generadas utilizando ECDSA (22) sobre la curva “secp256k1”, de las que se deriva la dirección pública de la cuenta de usuario. Cada transacción que un usuario realice en la cadena irá firmada con su clave privada para así garantizar el no repudio y la autenticidad del mensaje. Para poder suplantar la identidad de un usuario en la red, habría que descubrir la clave privada

asociada a la dirección de su cuenta. Esto implicaría resolver el problema del logaritmo discreto sobre una curva elíptica, que como ya se ha mencionado con anterioridad, es inabordable computacionalmente.

2.3.1. EVM

La Máquina Virtual de Ethereum (EVM) es el componente central que hace posible la funcionalidad de contratos inteligentes en la red Ethereum. Es un entorno de ejecución que permite a los desarrolladores ejecutar código en la Blockchain de Ethereum. Actúa como una máquina descentralizada que simula un ordenador completo, asegurando que los programas se ejecuten de manera consistente en todos los nodos de la red.

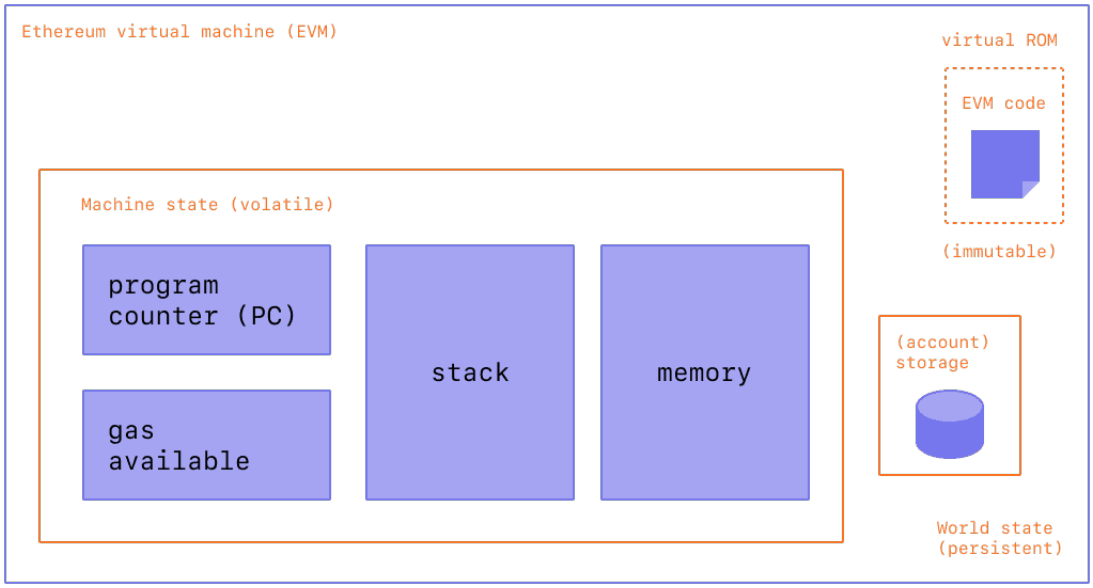


Figura 2.3: Diagrama de la EVM

Cadena privada

Debido a que el protocolo sobre el que se ha construido Ethereum es de código abierto, se puede replicar de manera privada. Esto permite tener un mayor control sobre la cadena y el mecanismo de consenso, así como sus activos y personalizar el protocolo para ajustarse a las aplicaciones específicas de la cadena.

2.3.2. Contratos inteligentes

Un contrato inteligente es un programa autoejecutable que se almacena en la cadena. Están escritos en lenguajes de programación, como Solidity, y se despliegan en la red para ejecutar transacciones y acuerdos de manera automática y sin intermediarios. La principal ventaja de los contratos inteligentes es que son inmutables y transparentes, lo que significa que, una vez implementados, no se pueden modificar y cualquier persona

puede verificar su código y funcionamiento. Esta transparencia permite prescindir de cualquier entidad que valide la ejecución del código, permitiendo generar aplicaciones descentralizadas basadas en esta tecnología.

Los contratos se desarrollan como se desarrollaría una clase en cualquier lenguaje orientado a objetos. Así es que cuando se despliega esta clase a la cadena, se crea una instancia con una dirección asociada para que los usuarios puedan interactuar con ella.

En la figura 2.3 se puede ver un diagrama de la EVM en la que se diferencian dos grandes bloques de memoria: “stack” y “memory”. En el primer almacén se guardarán las variables de instancia del contrato, mientas que en el segundo se guardarán todas aquellas variables creadas temporalmente en el interior de algún método.

Solidity

Solidity es un lenguaje de programación de alto nivel, muy similar a JavaScript (23), diseñado específicamente para crear contratos inteligentes que se ejecutan en la EVM.

En la figura 2.4 se pueden apreciar los distintos elementos que componen un contrato. En primer lugar, se definen una serie de variables de instancia, que como se mencionó antes se cargarán al stack de la máquina en el momento de la ejecución.

A continuación, se puede apreciar un bloque iniciado por la palabra “modifier”. Estas estructuras actúan como funciones que se puede especificar que se ejecuten antes de un método del contrato. En este caso accede a la variable global “msg”, que contiene información de la ejecución actual, para asegurar que es el propietario del contrato quien está llamando al método.

En el constructor se introduce por primera vez el uso de variables de memoria, asignadas de manera dinámica. Permite establecer las variables propias del contrato a la hora de instanciarlo.

Finalmente, se muestra un método que hace uso del modificador antes mencionado y accede a una variable de instancia para modificar su valor. La palabra “external” denota la visibilidad del método de cara a la cadena.

```
contract Election {
    struct Vote {
        string ballot;
        bool deleted;
        bool granted;
    }

    mapping(address => Vote) public votes_;
    address[] public votes_address_;
    uint256 candidates_count_;
    string public id_;
    address owner_;
    string public public_key_;

    modifier _ownerOnly() {
        require(msg.sender == owner_, "Only the owner can execute this method");
        _;
    }

    constructor(uint64 candidates_count, string memory id, string memory public_key) {
        owner_ = msg.sender;
        candidates_count_ = candidates_count;
        id_ = id;
        public_key_ = public_key;
    }

    function grant(address addr) external _ownerOnly {
        votes_[addr].granted = true;
    }
    ...
}
```

Figura 2.4: Estructura del contrato desarrollado

Yul

Se trata de un lenguaje intermedio entre Solidity y el bytecode ejecutado en la cadena. Resulta muy similar al ensamblador, permitiendo al programador una gestión total de la memoria con punteros y funciones como “mstore” y “mload” para cargar o guardar variables en memoria.

```
bytes memory public_key = bytes(public_key_);
uint256 public_key_size = public_key.length + 32;

bytes memory verify_input = new bytes(public_key_size + ballot_size + 64);
uint256 verify_input_size = verify_input.length + 32;

for (uint j = 32; j < public_key_size + 32; j += 32) {
    assembly {
        mstore(add(verify_input, j), mload(add(public_key, sub(j, 32))))
    }
}
```

Figura 2.5: Ejemplo de Yul

En este fragmento de código se copia el contenido de la variable "`public_key`" a "`verify_input`". El tamaño de palabra con el que operan las funciones yul es 32 bytes de y al principio de las variables del tipo "bytes" se reservan 32 bytes para almacenar el tamaño de la variable.

Contratos precompilados

Aunque el ensamblador utilizado por Ethereum es Turing completo (24), para realizar ciertas operaciones criptográficas como una función de hash, una suma de puntos en una curva elíptica, etc. puede resultar limitante sobre todo por el gas necesario para ejecutarlas. Para ello, junto con las instrucciones básicas del ensamblador se introdujo una serie de rutinas denominadas contratos precompilados que implementan estas funciones más complejas de ejecutar.

Estas rutinas están definidas en el propio código fuente del nodo que esté ejecutando la cadena, por lo que todos los nodos deben tener la misma implementación de estos para garantizar la integridad de la ejecución de cualquier contrato.

Para ejecutar estos precontratos se debe utilizar Yul y empaquetar todos los argumentos en un mismo bytearray. A continuación se muestra un ejemplo de ejecución del contrato "ecMul" para multiplicar un punto de una curva elíptica por un escalar.

```
function ecmul(uint256 x, uint256 y, uint256 scalar) public returns(uint256[2] p) {
    // With a public key (x, y), this computes p = scalar * (x, y).
    uint256[3] memory input;
    input[0] = bx;
    input[1] = by;
    input[2] = scalar;
    assembly {
        // call ecmul precompile
        if iszero(call(not(0), 0x07, 0, input, 0x60, p, 0x40)) {
          revert(0, 0)
      }
    }
}
```

Figura 2.6: Ejemplo llamada a contrato precompilado

En caso de fallar, la llamada devolverá un 0 y se deberá restablecer el estado interno de la EVM para evitar posteriores errores.

Capítulo 3

El protocolo

En este apartado se busca introducir formalmente el protocolo de votación sin entrar en los detalles de implementación.

3.1. Introducción al protocolo

Todo el protocolo se basa en la idea de desvincular el voto cifrado del usuario que realiza la acción. De esta manera se garantiza que, aún teniendo la clave de cifrado, no se puede asociar el contenido de una papeleta con un usuario.

Para lograrlo se deben distinguir tres entidades a la hora de votar: el votante, el servidor de autenticación y el servidor de almacenamiento público (Blockchain). Cuando un usuario quiera votar en una elección tendrá primero que negociar el acceso con el servidor de autenticación. En caso de que el usuario esté autorizado a votar, se le otorgará un ticket que le permita acceder al almacenamiento público usando unas credenciales efímeras, que son totalmente ajenas al primer servidor.

La clave radica en que tanto el ticket como las credenciales efímeras no se pueden asociar con la cuenta del usuario. En el siguiente apartado se profundizará para entender cómo se ha implementado este protocolo de manera teórica.

3.2. Descripción formal

Al comenzar una elección se deben crear las credenciales que utilizará el servidor para cifrar y firmar.

- Un administrador creará un par de claves usando ElGamal exponencial (G_{pk}, G_{sk}) y enviará al servidor la clave pública.
- El servidor genera un par de claves RSA (R_{pk}, R_{sk}) para esta elección.

3.2.1. Crear un voto

Crear credenciales

Antes de poder proceder se asume que el usuario tiene un certificado digital válido y se ha autenticado correctamente con el servidor.

- En el cliente, el usuario crea un par de claves secp256k1 que llamaremos (U_{pk}, U_{sk}). Se deberán guardar en el ordenador del usuario.
- Usando la clave pública del certificado digital del usuario se cifra (U_{sk}) y se guarda en el servidor asociado al usuario. En todo momento el usuario podrá recuperar su clave privada cifrada por si pierde su copia local.
- Usando U_{pk} el cliente cifrará U_{sk} y lo guardará también asociado al usuario. De esta manera en el futuro el usuario podrá verificar que el voto le pertenece al ser capaz de descifrar su clave privada.

Solicitar ticket

- El cliente ciega U_{pk} usando un número aleatorio y R_{pk}, y lo envía al servidor.

$$B(U_{pk}) = U_{pk} r^e \, mod \, N$$

- El servidor lo firma usando R_{sk} y guarda en la base de datos que se le ha emitido un ticket al usuario.

$$S(B(U_{pk})) = (U_{pk} r^e)^d \, mod \, N$$

- El cliente elimina la parte aleatoria aprovechando que $r^{ed} \equiv r$.

$$S(U_{pk}) = {U_{pk}}^d r r^{-1} \, mod \, N = {U_{pk}}^d \, mod \, N$$

- A través de un portal anónimo el usuario enviará $\{U_{pk}, \ S(U_{pk})\}$ para que el servidor le autorice a ejecutar el contrato y envíe a esa dirección el suficiente Ether para generar un voto. El servidor sabe que esta dirección está autorizada porque está firmada por él, pero no sabe de quién es. Este ticket se guardará en el servidor para asegurar que no se vuelva a utilizar.

Emitir voto

- El usuario crea un voto y lo cifra usando G_{pk}. Consistirá en un vector de 0 o 1 definiendo así afirmativo o negativo por cada candidato.

$$v = \{0, 0, 1\} \rightarrow v' = \{G_{pk}(r_0, v_0), ...\}$$

  En teoría, el usuario debería tener acceso a los escalares aleatorios utilizados para generar el voto, para así poder probar que su elección se ha cifrado correctamente.

De cara al recuento, para garantizar la integridad y corrección del voto, se generarán dos ZKPs: una verificando que cada elemento del vector es, o bien un 0 o un 1, y otra para verificar que sólo un elemento del vector tiene un 1.

La primera prueba de conocimiento nulo se conoce como prueba de anillo (25) y está compuesta por tantos anillos como candidatos haya. Para generarla se utilizará la componente aleatoria utilizada para construir el voto.

La otra prueba se basa en demostrar la igualdad de dos logaritmos discretos sobre distintos generadores a la que se le aplica la transformada de Fiat-Shamir (26) para prescindir de la componente interactiva de la prueba. Nuevamente se utilizará la componente aleatoria para generar esta prueba.

- El usuario ejecuta el contrato asociado con la elección directamente, usando su par de claves y guarda el voto junto con las dos pruebas generadas.

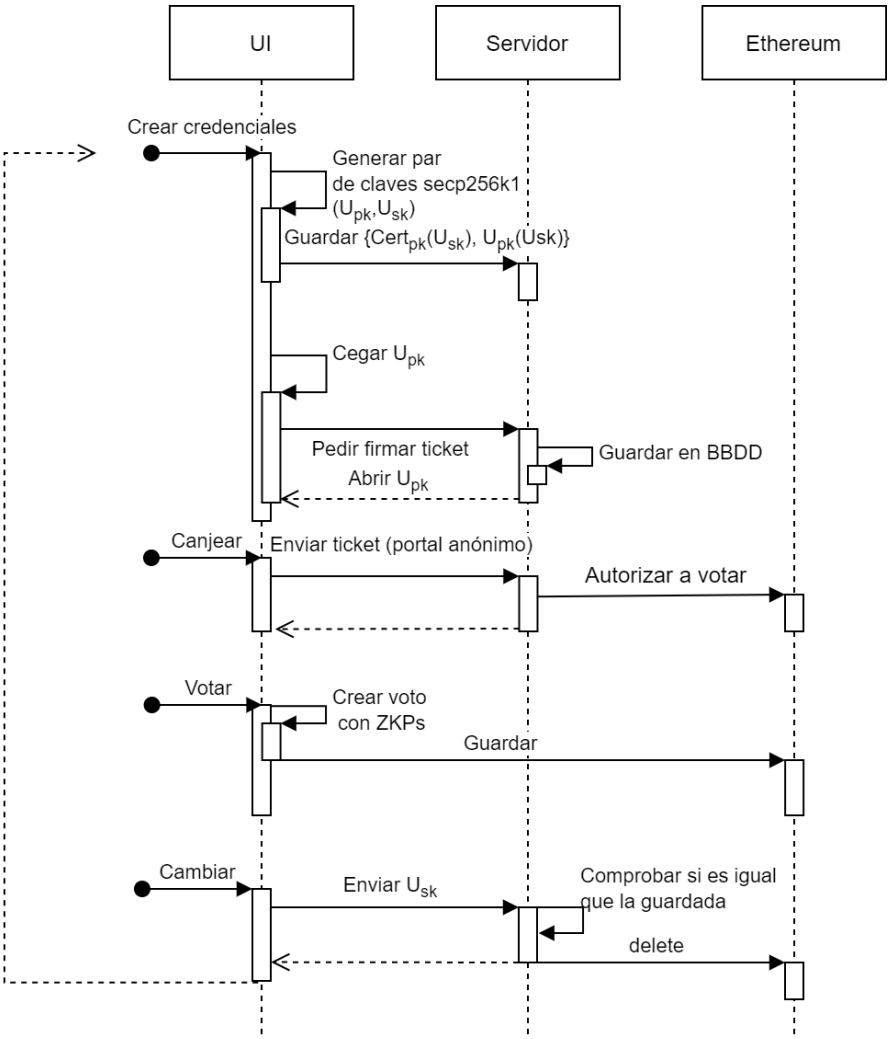


Figura 3.1: Diagrama de secuencia de crear y cambiar voto

3.2.2. Cambiar el voto

- El servidor pedirá al usuario la clave privada utilizada para interactuar con la cadena. A continuación, se usará para descrifrar la guardada en el servidor. En caso de fallar,

será que no es la pareja privada de la usada para cifrar inicialmente y se detendrá el proceso.

- El servidor ejecuta un contrato para eliminar el voto de la estructura de datos que lo contiene. Sólo el servidor es capaz de ejecutar este contrato porque en él se revisa la dirección desde la que se ejecuta.
- En el servidor se guarda un registro de esta operación asociando al usuario con la cuenta eliminada.
- El usuario deberá repetir por completo el proceso de creación de un voto.

3.2.3. Recuento

- En el momento en el que acabe la elección, el servidor ejecutará un método del contrato de la elección para que internamente en la cadena se verifiquen y sumen todos los votos cifrados. Para ello se utilizan una serie de contratos precompilados, capaces de realizar operaciones con los puntos de una curva elíptica.
- Un administrador pedirá el recuento al servidor y en el cliente introducirá la clave privada generada al principio de la elección para descifrar el resultado. Idealmente, esta clave estará dividida entre varios administradores o se forzará a la presencia de todos ellos para poder descifrar el resultado.

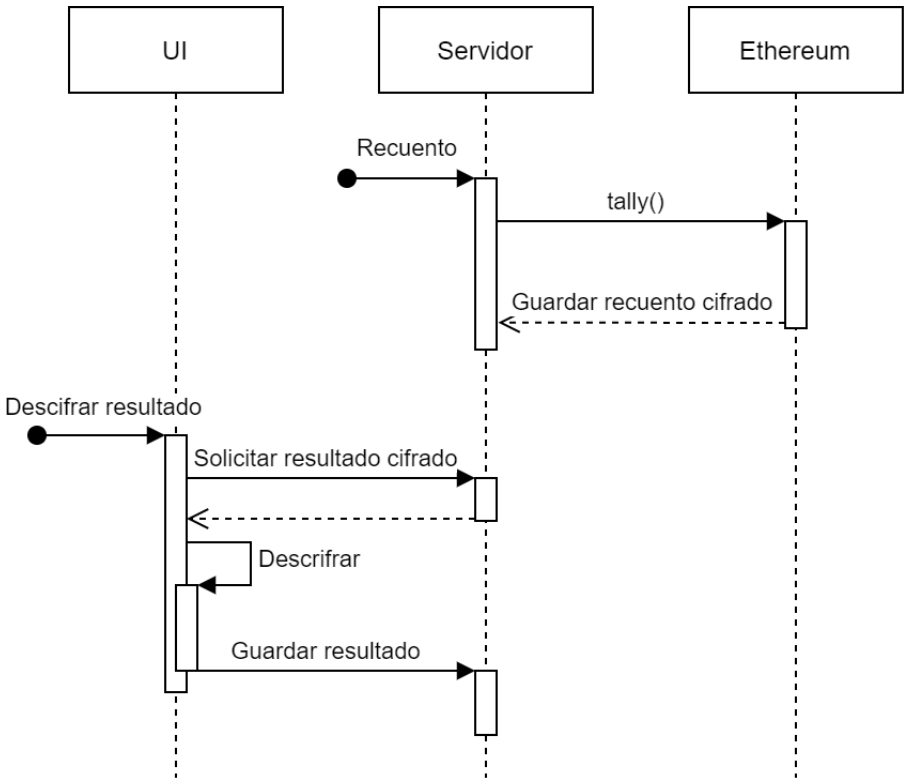


Figura 3.2: Diagrama de secuencia del recuento

3.2.4. Auditoría externa

El protocolo garantiza que todas las operaciones se realizan de manera transparente al usuario, ya sea porque se realizan en el cliente o porque se realizan en la cadena. No obstante, el hecho de desvincular al usuario de la papeleta plantea una nueva problemática: se puede añadir votos a la cadena que no estén asociados a ningún votante. Esto solo lo podría realizar el propietario del contrato, por lo que hay que depositar un alto grado de confianza en la administración de la votación.

Este problema se podría detectar en el momento de emitir el voto a la cadena, comprobando que el número de votos emitidos en el servidor sea mayor que el número de cuentas autorizadas a votar en la cadena. Por desgracia, esta comprobación no se puede hacer en la cadena ya que no se tiene la información de los votos emitidos en el servidor.

La opción sería hacer, o bien comprobaciones periódicas de que la condición descrita se cumple, o auditar este proceso de manera externa. La entidad auditora tendría acceso al listado de votos emitidos en tiempo real y crearía un servicio que se suscriba a un evento emitido cada vez que se autorice una cuenta en la cadena. De esta manera se podría detectar cualquier anomalía inmediatamente. De hecho, se podría hacer pública la lista de votos emitidos para que así cualquier usuario pudiera comprobar la integridad del recuento.

En caso de implementarse esta solución, en el listado de votos emitidos se debería guardar algo que asegure que ese registro pertenece a un usuario autorizado y no ha sido creado ficticiamente por el servidor. Esto se puede conseguir fácilmente usando el certificado individual de cada usuario, que firmaría por ejemplo, el resto de campos en este registro.

Con esta solución, para que el servidor pudiera emitir un voto tendría que poseer un certificado de cliente por lo que sería necesario que tanto la administración de la votación como la Autoridad Certificadora fuesen corruptas.

Capítulo 4

Implementación

A continuación se describirá cómo se ha implementado este protocolo, así como las diferencias respecto de la definición formal.

4.1. Motor criptográfico

Toda la lógica de cifrado, tanto de los votos como de los tickets, ha sido desarrollada en Rust (27) utilizando una serie de librerías: elastic-elgamal (28), blind-rsa-signatures (29), ecies (30) y rsa (31). Estas librerías ofrecen una interfaz para trabajar con ambos cifrados de una manera muy cómoda y rápida. El problema es que en el caso de elastic-elgamal la librería restringe el acceso a las componentes aleatorias utilizadas para generar los cifrados por lo que solo se pueden producir las ZKP ya implementadas en la librería.

Para acceder a estas funciones desde la web se ha utilizado WebAssembly (32). Se trata de un lenguaje de bajo nivel muy similar al ensamblador que ofrece rendimiento nativo en la web. El ecosistema de Rust está ampliamente integrado con WebAssembly por lo que simplemente hace falta utilizar una utilidad de Rust, denominada wasm-pack (33) para generar este código.

La cadena de bloques se ha ejecutado utilizando la propia librería desarrollada por Ethereum para ejecutar un nodo de la cadena: Geth (34). A pesar de existir utilidades que permiten la compilación cruzada con Rust como FFI (35), se tuvo que compilar por separado. Esto se debe a que el código fuente de Geth cuenta con su propio proceso de compilación, lo que dificulta añadir una librería compilada. La conexión se realizó por tanto, ejecutando directamente la librería desarrollada Rust como un subproceso (ver A.3 y A.2).

4.2. La base de datos

Aunque los votos se almacenen en la cadena de bloques, toda la demás información debe ser persistida en disco. Para ello se ha utilizado la base de datos relacional PostgreSQL (36). En la figura 4.1 se representan las distintas entidades y cómo se relacionan.

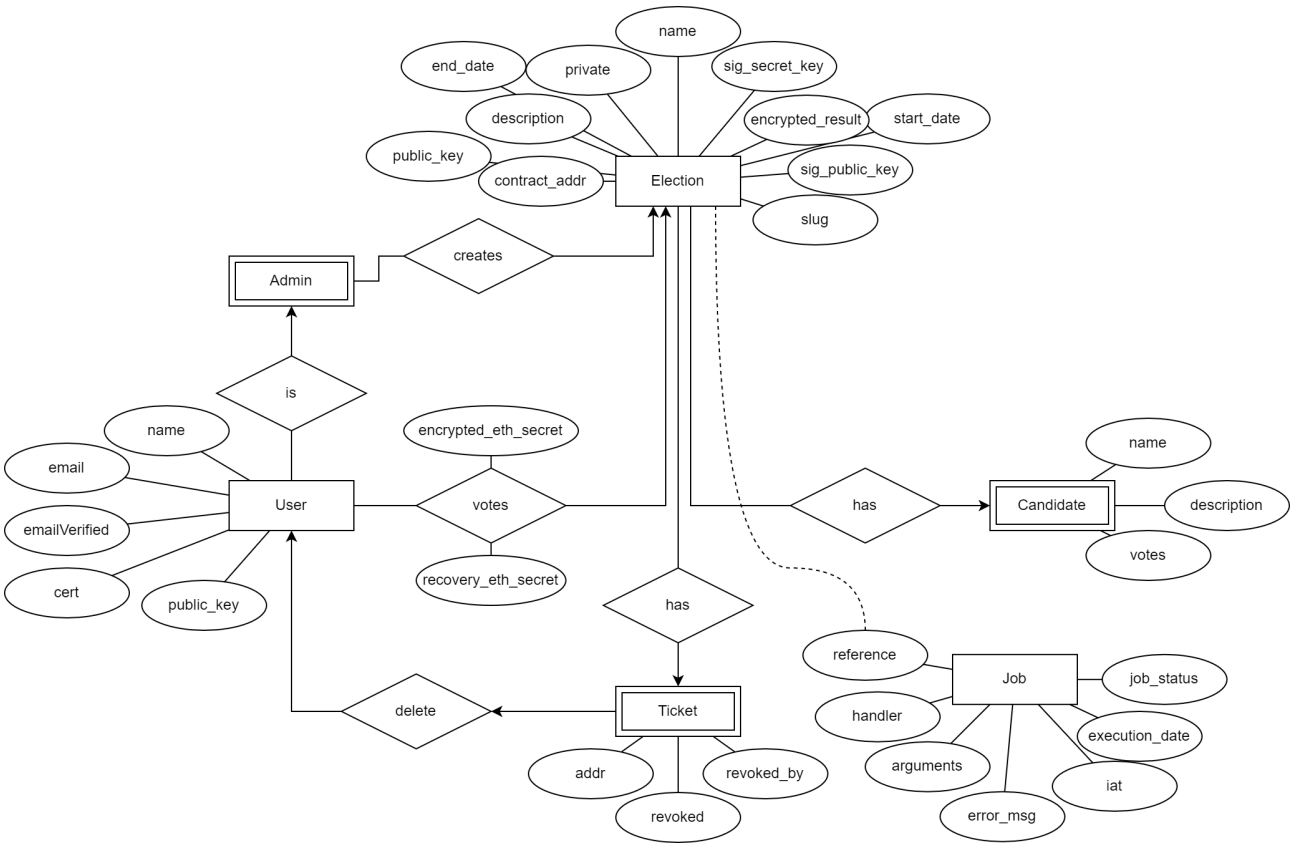


Figura 4.1: Diagrama entidad-relación de la base de datos

La entidad “Jobs” no está directamente relacionada con ninguna otra entidad, puesto que no aporta información adicional. Esta tabla representa tareas programadas que se ejecutarán en segundo plano y se relacionan de una manera no directa con la tabla a la que afectarán. En el diagrama se puede ver como esta relación indirecta realmente solo existe con la entidad “Election’,’ puesto que no se han implementado tareas que afecten otras tablas.

4.3. La web

Como se ha mencionado anteriormente, para el desarrollo de la web se utilizó el framework NextJS. A continuación, en la figura 4.2 se muestra la estructura de directorios utilizada para el proyecto.

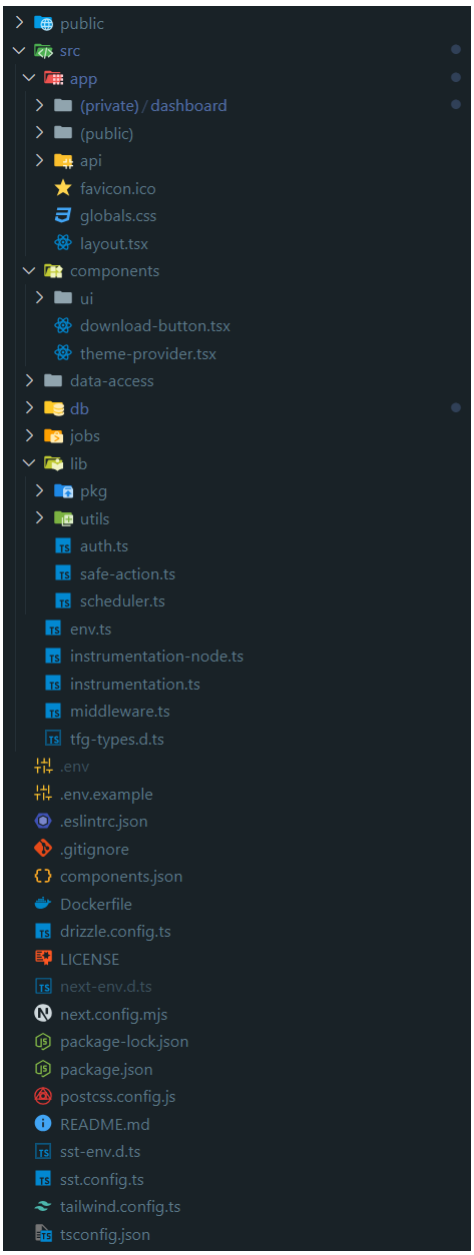


Figura 4.2: Estructura de la web

- **`@/public`.** Archivos públicos para los usuarios como imágenes o el contrato y su bytecode.
- **`@/app`.** Rutas de la web. Las subcarpetas public y private denotan las partes en las que se requiere autenticación o no.
- **`@/components`.** Componentes compartidos entre distintas rutas de la web. En el directorio `@/components/ui` se encuentran los que constituyen la interfaz desarrollados por shadcn/ui (37).
- **`@/data-access`.** Funciones para el acceso a la base de datos.
- **`@/db`.** Configuración de la base de datos, representación del schema y funciones de conexión.
- **`@/jobs`.** Tareas que se ejecutarán en segundo plano por el scheduler.

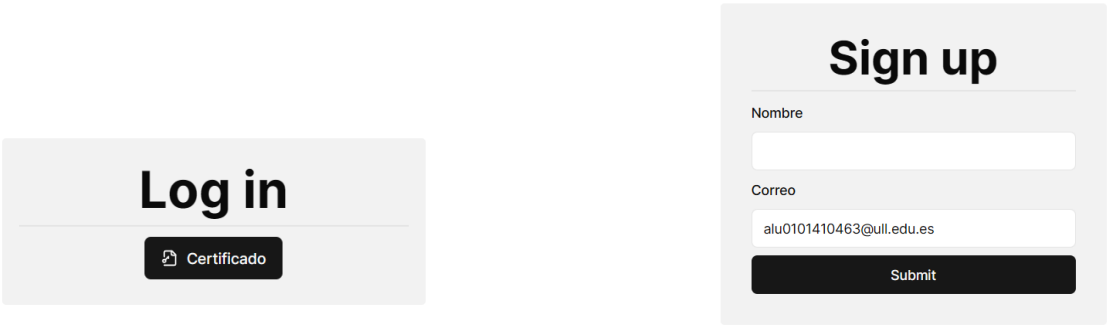
- **`@/lib`.** Archivos de configuración de algunas librerías además de otras desarrolladas para el proyecto y algunas funciones compartidas entre todo el proyecto en `@/lib/utils`. En `@/lib/pkg` se encuentran las librerías compiladas del motor criptográfico.

4.3.1. Inicio de sesión y registro

Como ya se especificó en la definición del protocolo, es necesario que todo usuario tenga un certificado asociado. Para obtenerlo, en el inicio de sesión, se utiliza el handshake SSL (38), especificando que se requiera un certificado de cliente válido.

Esto se ha logrado utilizando dos dominios para la web:

- `e3vote.iaas.ull.es`. Accepta todo el tráfico de la web sin requerir que el usuario presente un certificado válido. La ruta `/api/request-auth` será redireccionada al otro dominio.
- `login-e3vote.iaas.ull.es`. Acepta solo el tráfico de `/api/request-auth` solicitando un certificado válido por parte del cliente. El resto del tráfico será redireccionado al otro dominio.



(a) Login

(b) Register

Dado que NextJS no tiene acceso al handshake para extraer la información, se ha configurado Nginx (39) como un proxy reverso que maneje los certificados y los devuelva como cabeceras (X-SSL-CERT y X-SSL-DN).

```
server {
        listen 443 ssl;

        server_name login-e3vote.iaas.ull.es;

        ssl_protocols       TLSv1.1 TLSv1.2;
        ssl_ciphers         HIGH:!aNULL:!MD5;
        ssl_session_cache   shared:SSL:10m;
        ssl_session_timeout 10m;

        ssl_certificate         /etc/tfg/certs/e3vote.iaas.ull.es.crt;
        ssl_certificate_key     /etc/tfg/certs/e3vote.iaas.ull.es.key;
        ssl_client_certificate  /etc/tfg/certs/TFG-RootCA.crt;

        ssl_verify_client on;

        location /api/request-auth {
                proxy_set_header   Host              $host;
                proxy_set_header   X-Real-IP         $remote_addr;
                proxy_set_header   X-Forwarded-Proto $scheme;
                proxy_set_header   X-Forwarded-For   $proxy_add_x_forwarded_for;

                proxy_set_header   X-SSL-CERT        $ssl_client_escaped_cert;
                proxy_set_header   X-SSL-DN          $ssl_client_s_dn;

                proxy_pass http://localhost:3000;
        }

        location / {
                return 301 https://e3vote.iaas.ull.es$request_uri;
        }
}
```

Figura 4.4: Configuración Nginx del vhost login-e3vote

De esta manera, al cargar el formulario de registro se hará una petición al endpoint `/api/request-auth`, que devuelve el certificado y los datos del mismo como respuesta utilizando json. En caso de haber algún problema con el certificado cliente se detendrá el proceso y se lanzará un error.

```
{
  "cert": "-----BEGIN%20CERTIFICATE-----%0[...]%0A-----END%20CERTIFICATE-----%0A",
  "email": "alu0101410463@ull.edu.es"
}
```

Figura 4.5: Respuesta de `/api/request-auth`

Se puede ver que, junto con el certificado, se envía el email del usuario que estaba guardado en el mismo. Este será el que se use en el registro sin permitir al usuario que lo cambie.

4.3.2. Emitir un voto

El layout compartido de la ruta `/vote/[slug]` comprueba que un usuario pueda ejercer el voto en una votación y que esta esté todavía abierta. Si no tiene permisos, se le redirigirá al inicio y, en caso de que la votación no esté abierta, solo se renderizará la información de la misma.

```
const election = await getElectionBySlug(params.slug);

if (!election) {
  return <div>Election not found</div>;
}

let canVote = !election.isPrivate;
const session = await auth();

if (election.isPrivate) {
  canVote = await isAuthorizedToVote(session?.user.userId!, election.id);
}

if (!canVote) {
  redirect('/');
}
```

Figura 4.6: Comprobaciones en el layout compartido

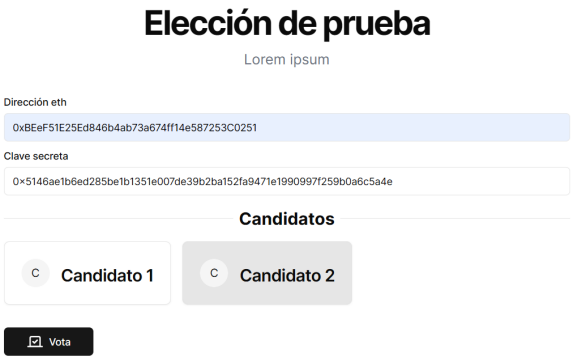
Pasos previos

En esta sección el usuario generará su par de claves para interactuar con la cadena y el ticket para recibir la autorización en esa cuenta. Existe un botón para poder descargar toda esta información en un archivo json. El código QR tiene el mismo efecto que canjear el ticket manualmente, pero, al hacerlo con el móvil se garantiza que no se accede con la misma IP, para que así el servidor no pueda relacionar el ticket con el usuario.

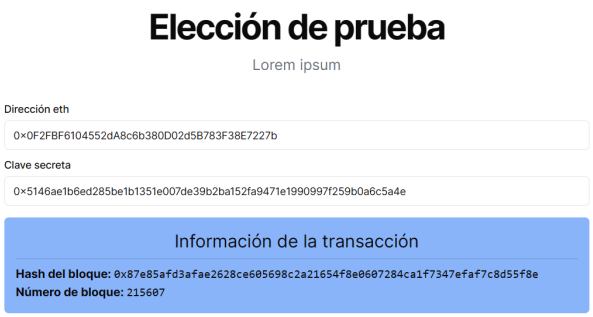
Figura 4.7: Pasos previos

Seleccionar candidato

Una vez el usuario ha adquirido sus credenciales y canjeado el ticket puede proceder a votar. Para ello debe especificar su dirección y su clave secreta. Una vez realizada la transferencia se mostrará el número de bloque en el que se ha incluído, así como el hash del mismo.



(a) Seleccionar candidato



(b) Resultado del voto

Eliminar voto

Una vez emitido un voto, el usuario debe ser capaz de borrarlo en caso de cambiar de opinión. Para ello simplemente tendrá que facilitar la clave secreta y en el servidor se comprobará que coincida con la guardada cifrada con la componente pública.

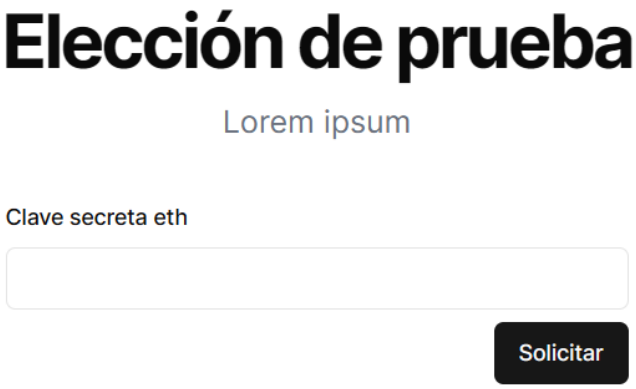


Figura 4.9: Eliminar voto

Recuperar credenciales

Se puede dar el caso que un usuario pierda sus credenciales y quiera modificar su voto. Por eso, en los pasos previos se guarda en el servidor U_{sk} cifrado con la clave pública del certificado digital del usuario.

El usuario deberá adjuntar su clave privada para descifrar U_{sk} en el navegador. Aunque esta solución se ha implementado en el prototipo, dificulta mucho la accesibilidad la página. Un usuario tiene que tener su certificado en formato `PKCS12` (40) para poder importarlo en el navegador y también debe tener la clave privada del certificado en formato `PKCS1` (41) para poder utilizarlo en esta ruta.

Figura 4.10: Recuperar cuenta

4.3.3. Presentar ticket

Para transferir los fondos necesarios para votar a una cuenta de Ethereum se debe utilizar el ticket obtenido en `/vote/[slug]/previous`. Este está codificado en base64 por comodidad, pero internamente (figura 4.12), el ticket guarda la dirección a la que enviar el Ether, la votación en la que quiere participar y la firma del mensaje.

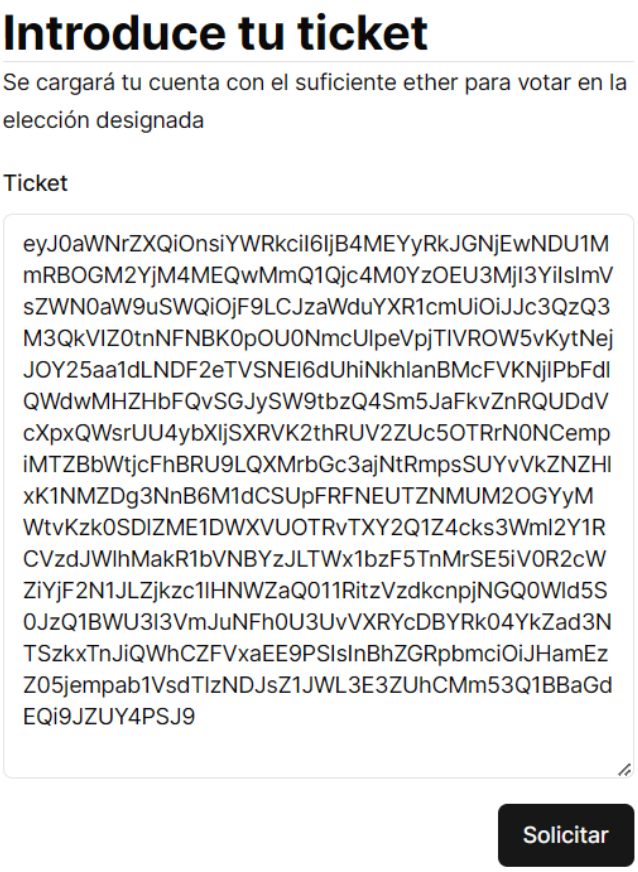


Figura 4.11: Faucet

```
{
  "ticket": {
    "addr": "0x845A3A4f18AF3667E1cecF7d7f1a218C5F8fa8b4",
    "electionId": 25
  },
  "signature":"gKS/OwpxPO7c+NU1XixmqEQdhWYSwPaQPfnWngi5QlDbc+B+Lmy[...]",
  "padding":"3Z+6NDIbLLRvjQzWvZd8NpOq3dDku3Fx+Kj/ALMLsro="
 }
```

Figura 4.12: Ticket decodificado

Primero se creará un voto falso para calcular cuanto gas va a requerir el usuario a la hora de votar llamando al método del contrato encargado de ello (figura 4.13). Después, se ejecutará el método "`grant`" del contrato que almacena en el mapping "`votes_`" que la dirección está autorizada a votar (figura 4.14).

Finalmente se guardará en la base de datos que la dirección contenida en el ticket ha sido autorizada a votar para así evitar votos dobles.

```
const electionContract = new web3.eth.Contract(abi, contractAddr);

const ballot = encryptVote(publicKey, 0, candidateCount);

const gas = await electionContract.methods
  .vote(ballot)
  .estimateGas({ from: clientAddr });
```

Figura 4.13: Calcular coste de un voto

```
const encodedABI = electionContract.methods.grant(clientAddr).encodeABI();

const signed = await web3.eth.accounts.signTransaction(
  {
    from: env.ETH_ACCOUNT,
    to: contractAddr,
    data: encodedABI,
    gasPrice,
    gas,
  },
  env.ETH_PRIV
);
await web3.eth.sendSignedTransaction(signed.rawTransaction);
```

Figura 4.14: Autorizar una cuenta ethereum a votar

4.3.4. Panel de control

En esta sección de la página los administradores podrán crear y visualizar elecciones. En todas las subrutas del panel de control hay un botón para añadir nuevas elecciones que despliega el menú lateral (figura 4.17).

Entre la información que se muestra en esta página, de cada votación se incluye su estado, que puede variar entre open (si se puede votar), closed (si ya se ha realizado el escrutinio) o upcoming (todavía no ha empezado).

Cuando se seleccione una votación se cargará toda la información de ella usando el slug de la ruta una vez y, aprovechando el sistema de layouts de NextJS, no será necesario requerir esa información de nuevo mientras se navegue dentro del dashboard. Este comportamiento se puede lograr usando la API Context de React (42) que permite compartir estado con componentes hijos.

```tsx
export default async function RootLayout({
  children,
  params,
}: Readonly<{
  children: React.ReactNode;
  params: { slug: string };
}>) {
  const election = await getElectionBySlug(params.slug);

  if (!election) {
    return <div>Not found</div>;
  }

  const candidates = await getCandidates(election.id);

  return (
    <div className="grid lg:grid-cols-[280px_1fr]">
      <ContextProvider value={{ ...election, candidates }}>
      ...
```

Figura 4.15: Contexto compartido

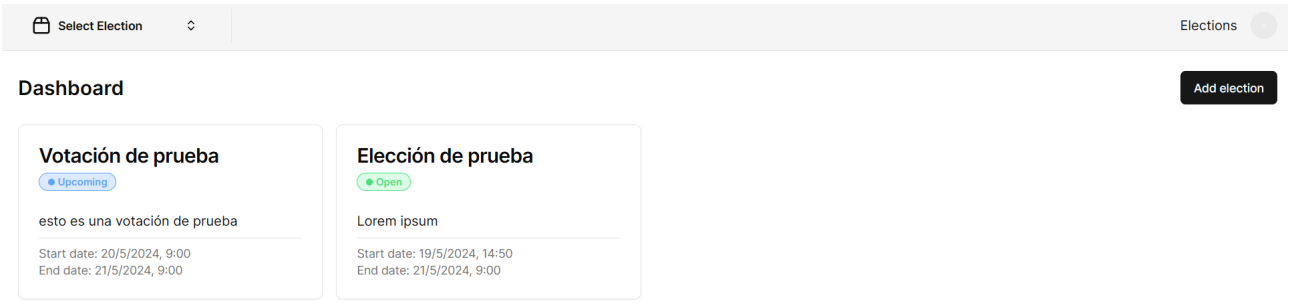


Figura 4.16: Panel de control

A la hora de crear una votación, se rellenará el formulario con la información requerida y una vez enviada la solicitud, se mostrará en el cliente el par de claves ElGamal que se utilizarán para cifrar los votos.

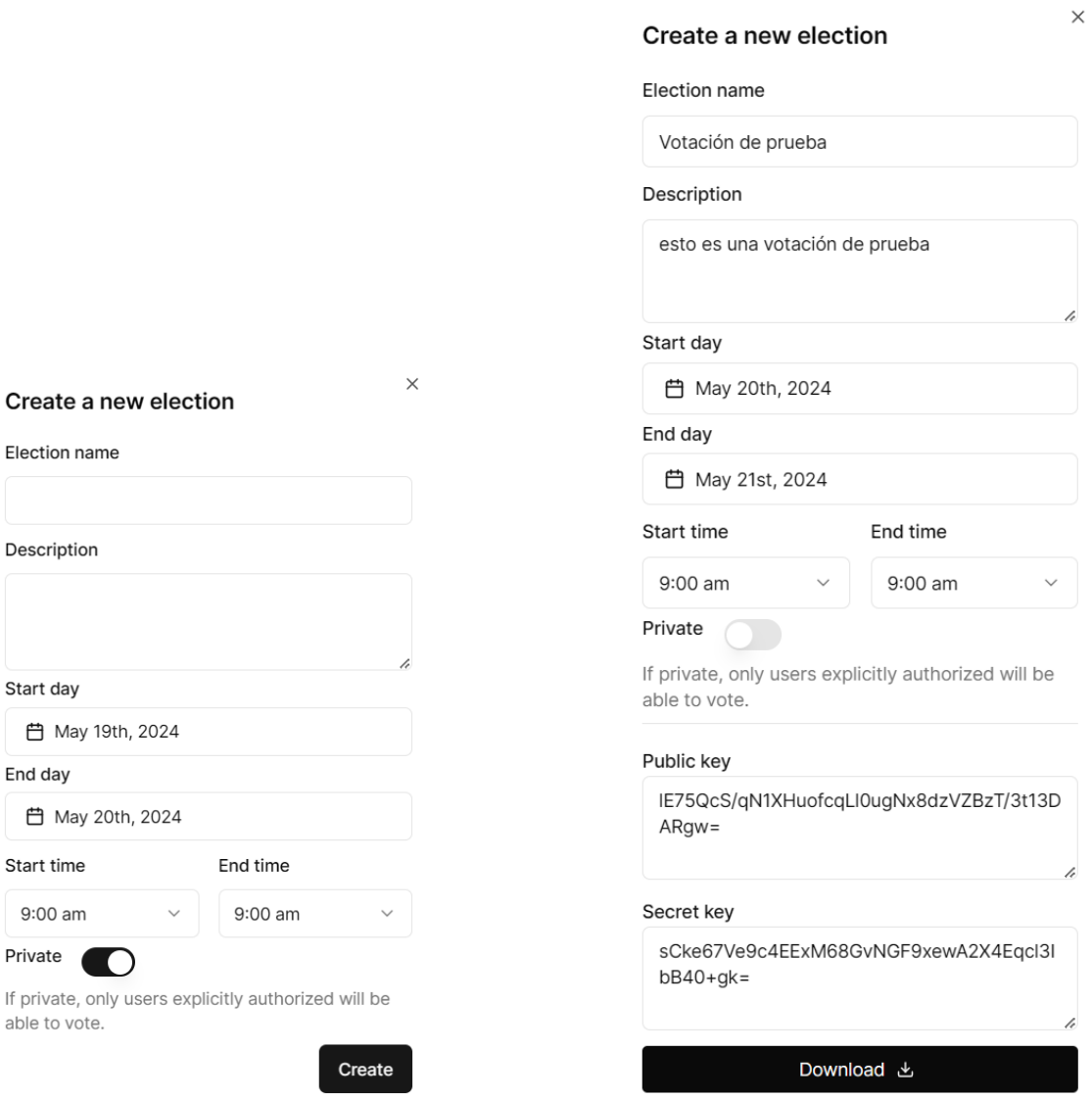


Figura 4.17: Formulario para crear una votación

A continuación se muestran las distintas subrutas que existen por cada votación dentro del dashboard.

Candidatos

Esta página muestra cada uno de los candidatos para una votación, dando la opción de añadir más o eliminar alguno. Esta edición será bloqueada en el momento que empiece la votación. Importante mencionar que este bloqueo no solo se realiza en la interfaz, sino que el servidor también bloqueará peticiones que modifiquen la votación.

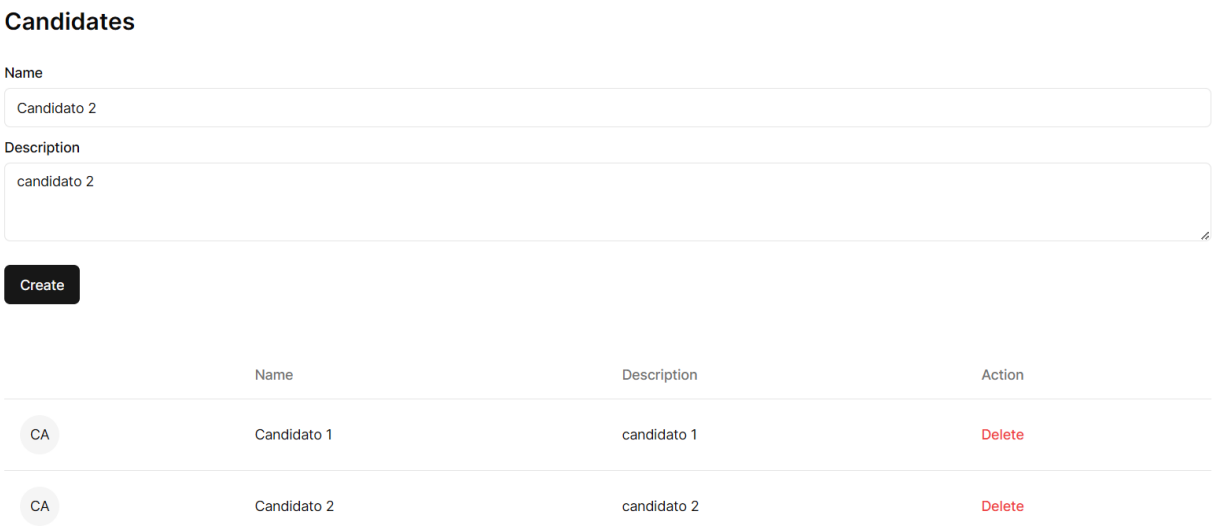


Figura 4.18: Candidatos

Votantes

En caso de crear la votación como privada, se podrá gestionar quién tiene acceso a la misma. Para ello se debe introducir el email de la persona y, en caso de que no esté dada de alta en el sistema todavía, se le creará un nuevo registro únicamente con su correo. Posteriormente, el usuario tendrá que cumplimentar el resto de información.

Si por el contrario la elección es pública, en esta página solo se verá aquellas personas que ya han votado.

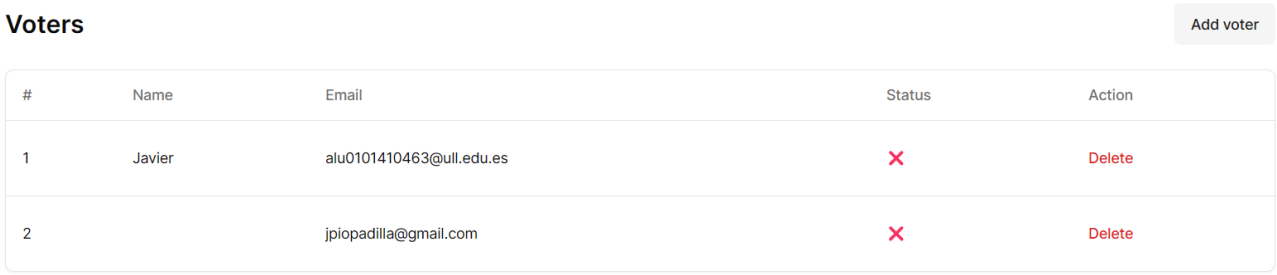


Figura 4.19: Votantes

Resultados

Una vez finalizada la votación, se realizará el escrutinio y se guardará en la base de datos el resultado cifrado. En esta página se puede forzar el escrutinio, lo que finalizaría la votación antes de tiempo.

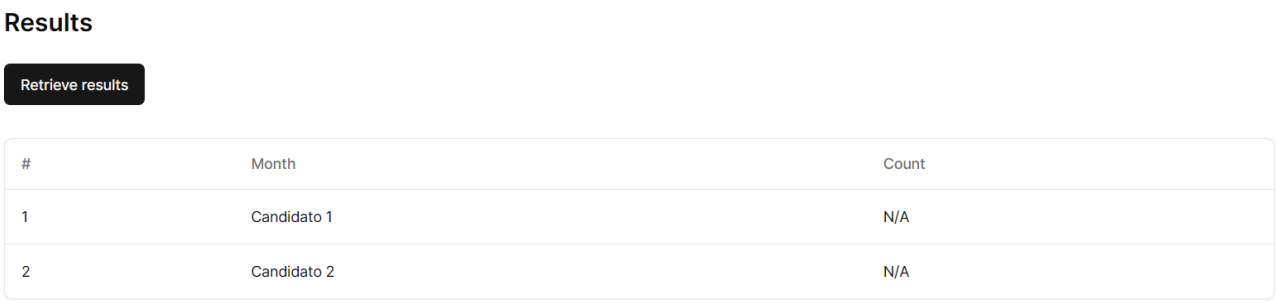


Figura 4.20: Pendiente de escrutinio

El descifrado del resultado también se hará a través de esta página: el administrador proporcionará la clave privada y en el cliente se descifrarán los resultados, para posteriormente guardarlos en claro en el servidor. Importante destacar que en ningún momento se enviará la clave privada al servidor.

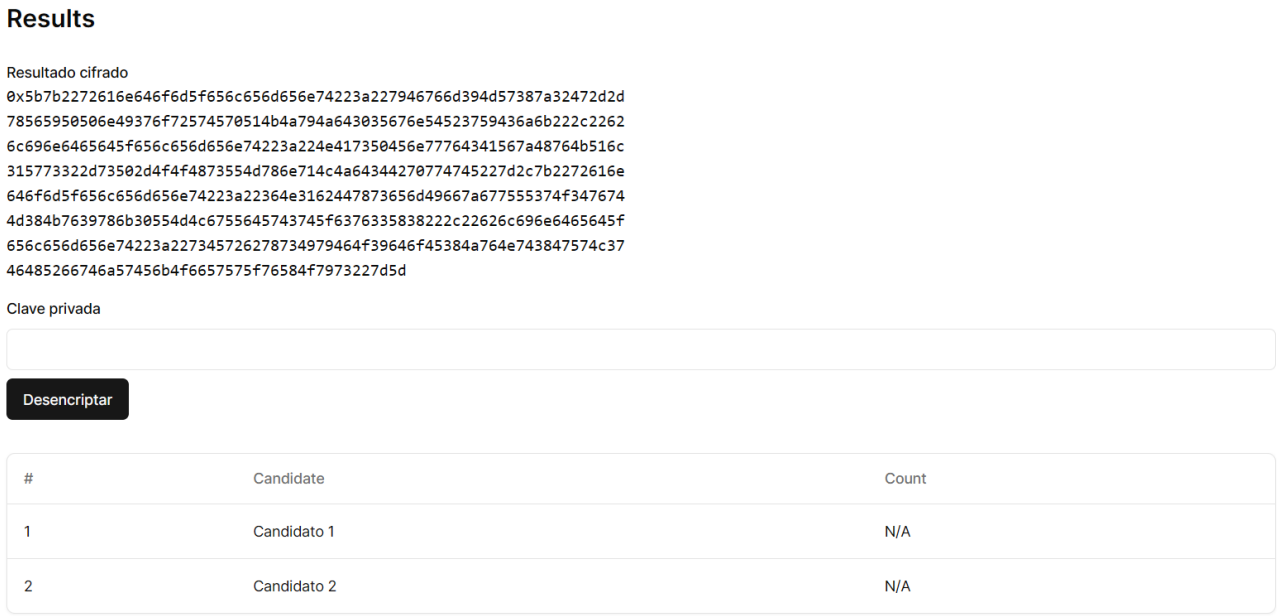


Figura 4.21: Resultado cifrado

4.4. Blockchain

Para poder trabajar de manera local en un entorno de desarrollo se ha creado una red básica con tres nodos: uno para firmar bloques, uno para hacer consultas a través de http y uno como bootnode, que sirve de intermediario para establecer conexiones entre los demás nodos. Para ejecutar cada nodo, como ya se mencionó anteriormente, se ha utilizado Geth, la implementación oficial del protocolo Ethereum en Go (43).

Como se trata de una red de desarrollo, el mecanismo de consenso (44) elegido ha sido PoA (Proof of Authority), en el que se designa qué nodos están autorizados a validar bloques. Resulta bastante sencillo y mucho más eficiente computacionalmente que otros como PoW (Proof of Work).

Debido al mecanismo de consenso elegido, la cadena tuvo que ser desactualizada hasta el hard fork (45) de Londres, lo que limita la versión de Solidity también. Esto no supone gran problema, puesto que las mejoras implementadas en el lenguaje están destinadas a optimizar los contratos de cara al gas; teniendo en cuenta que el gas de esta cadena es ficticio, no resulta ser un inconveniente.

La restricción de acceso de la librería elastic-elgamal mencionada anteriormente, impide utilizar los contratos preexistentes para operar con curvas elípticas. Debido a que la librería tampoco permite acceder a las coordenadas de los puntos de la curva, los votos generados se deben serializar para poder trabajar fuera de Rust. Por ello, se ha modificado el código fuente de Geth para incluir dos nuevos contratos precompilados que permitan verificar las ZKPs de un voto y sumarlo a un acumulador.

El contrato “Election”

En la figura 2.4 se pueden apreciar los distintos atributos (variables de instancia) que tiene el contrato desarrollado. A la hora de crearlo se debe especificar el número de candidatos (para poder saber el tamaño del acumulador) y la clave pública para trabajar con las ZKPs.

Como ya se comentó anteriormente, para llamar a los contratos precompilados (ver A.1) se debe agrupar todos los argumentos que éste reciba en un solo bytearray. Esto se ha logrado organizando la memoria como se ve en la figura 4.22, utilizando operaciones a bajo nivel con Yul.

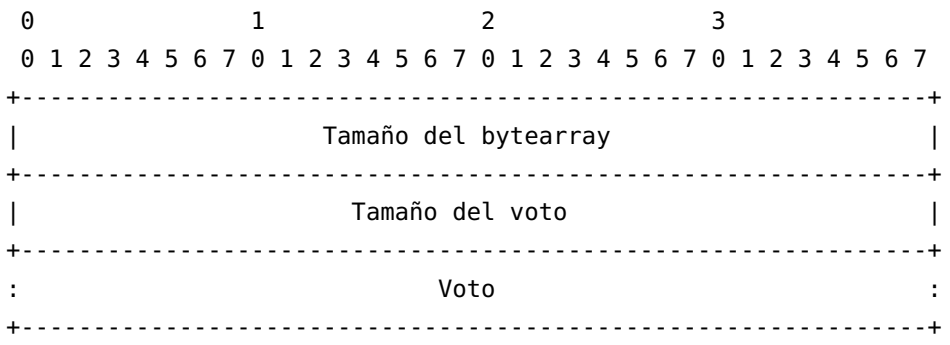


Figura 4.22: Estructura de argumentos para la verificación de un voto

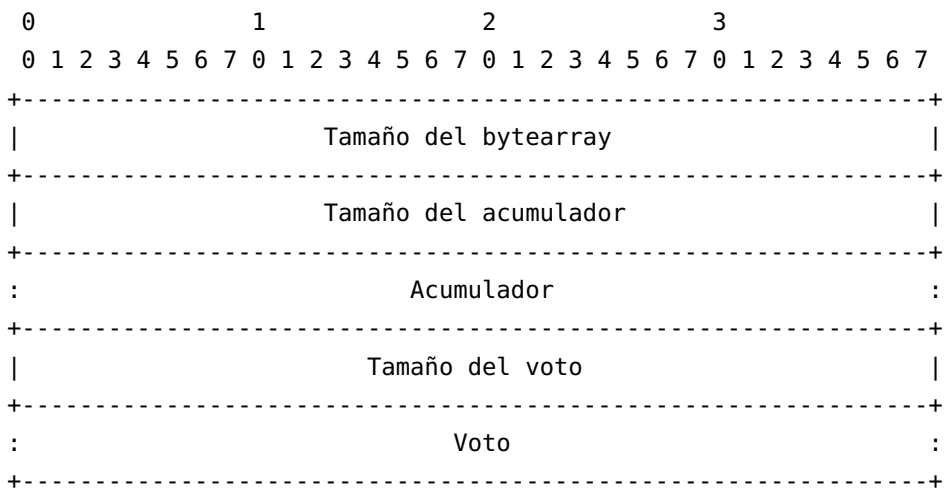


Figura 4.23: Estructura de argumentos para la suma de un voto y un acumulador

El contrato encargado de sumar los votos recibe un voto y un acumulador que es el que finalmente será el resultado de la votación. Este acumulador inicialmente no existirá, por lo que el contrato verifica que el acumulador esté bien serializado y, en caso de ser nulo generará uno con todos los candidatos a cero (ver A.3).

Capítulo 5

Conclusiones y líneas futuras

El avance de la tecnología y la seguridad informática hacen cada día más posible implementar un protocolo de votación online, que permita a todos los usuarios votar de manera cómoda y segura, garantizando que su voto es escrutado correctamente. En este trabajo se ha estudiado e implementado una de las posibles soluciones a los retos que propone una votación segura a través de internet y, aunque no está exento de ataques informáticos y posibles vulnerabilidades, plantea el uso de Blockchain para facilitar la confianza de manera distribuida en el sistema.

A pesar del resultado del trabajo, no podemos confirmar que se pueda implementar de manera totalmente segura un sistema de votación online a gran escala, al menos por ahora. Se trata de una infraestructura crítica para cualquier país que, aunque actualmente resulte algo arcaico, sus problemas están bien estudiados y son pocas las vulnerabilidades que presenta.

El estado actual del trabajo plantea tres mejoras claras para el futuro:

1. Implementar un control de emisión de votos fraudulentos por parte del servidor. Ya se ha discutido esta problemática ofreciendo una solución en el apartado de la definición formal del protocolo. No se pudo implementar por falta de tiempo. Esto se debe a que habría supuesto migrar toda la aplicación a una versión de escritorio para facilitar la gestión de los certificados. El desarrollo de esta solución plantea el mismo problema que recuperar un voto: aunque utilizando SSL se puede acceder fácilmente a la componente pública del mismo, la privada es totalmente inaccesible. En caso de ser una versión de escritorio se podría solicitar total acceso al certificado al usuario.

2. Optimizar operaciones criptográficas. Las librerías utilizadas en este trabajo se eligieron por la comodidad que ofrecían y no tanto por lo seguras que fueran. En una implementación real, no son viables. Además, Ethereum implementa de forma nativa la multiplicación y suma de puntos sobre una curva elíptica, lo que permitiría prescindir de los contratos precompilados desarrollados para este trabajo.

3. Implementar verificación del email. Este paso de seguridad extra garantiza que el propietario del certificado es quien dice ser, debido a que tiene acceso también a la cuenta que en él reside. No se ha implementado, dado que habría supuesto

configurar y desplegar también un servidor SMTP (46) que pudiera enviar correos a los usuarios.

Capítulo 6

Summary and Conclusions

Advances in technology and computer security are making it increasingly possible to implement an online voting protocol that allows all users to vote comfortably and securely, ensuring that their vote is counted correctly. This work has studied one of the possible solutions to the challenges of voting over the Internet and, although it is not exempt from computer attacks and possible vulnerabilities, it proposes the use of Blockchain to facilitate trust in the system.

Despite the outcome of the work, we do not consider that a large-scale online voting system can be securely implemented, at least for now. This is a critical infrastructure for any country that, although currently somewhat archaic, its problems are well studied and there are few vulnerabilities.

The current state of the project poses three clear improvements:

1. Implement a control of fraudulent votes cast by the server. This problem has already been discussed and a solution has been offered in the section on the formal definition of the protocol. It could not be implemented due to lack of time. This is because it would have meant migrating the entire application to a desktop version to facilitate the management of certificates. The development of this solution has the same problem as retrieving a vote: although the public component of the vote is easily accessible using SSL, the private component is totally inaccessible. In the case of a desktop version, full access to the certificate could be requested to the user.

2. Optimize cryptographic operations. The libraries used in this work were chosen for the convenience they offered and not so much for their security. In a real implementation, the libraries used are not viable. In addition, Ethereum natively implements point multiplication and addition on an elliptic curve, which would allow to not use the developed precompiled contracts.

3. Implement email verification. This extra security step ensures that the owner of the certificate is who he claims to be since he also has access to the account that resides in it. It has not been implemented since it would have meant configuring and deploying also an SMTP server that could send emails to users.

Capítulo 7

Presupuesto

En este capítulo se expone una estimación del presupuesto del proyecto. La tabla 7.1 recoge cada una de las tareas realizadas junto con el coste asociado.

Tarea	Precio €	Horas	Total
Investigación criptografía homomórfica	25	20	500€
Investigación Ethereum	25	50	1.250€
Desarrollo interfaz web	30	80	2.400€
Desarrollo de Contratos inteligentes	30	120	3.600€
Integración blockchain e interfaz web	30	80	2.400€
Total		360	10.150€

Tabla 7.1: Desglose de tareas

A continuación, se muestra la estimación de ejecutar cada servicio durante un día en una máquina virtual (47) de AWS (48) ubicada en Irlanda (eu-west-1). En lo referido a la cadena de bloques no se puede estimar fácilmente puesto que dependerá de la carga y el número de nodos. Además, como se busca que la cadena la conforme un grupo heterogéneo sin regulación, no habrá un estándar para el nodo de cómputo.

Recurso	Precio €
Instancia EC2 c6g.xlarge web	3,5016€
Instancia EC2 r3.xlarge base de datos	8,904€
Total	12,4056€

Tabla 7.2: Servicios

Apéndice A

Contratos

A.1. Contrato Election

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract Election {
    struct Vote {
        string ballot;
        bool deleted;
        bool granted;
    }

    mapping(address => Vote) public votes_;
    address[] public votes_address_;
    address[] public deleted_votes_address_;
    uint256 candidates_count_;
    string public id_;
    address owner_;
    bool public ended_ = false;
    bytes public result_;
    string public public_key_;

    modifier _ownerOnly() {
        require(msg.sender == owner_, "Only the owner can execute this method");
        _;
    }

    modifier _notEndend() {
        require(ended_ == false, "Election has ended");
        _;
    }

    constructor(uint64 candidates_count, string memory id, string memory public_key) {
        owner_ = msg.sender;
```

```
33          candidates_count_ = candidates_count;
34          id_ = id;
35          public_key_ = public_key;
36      }
37
38      function grant(address addr) external _notEndend _ownerOnly {
39          votes_[addr].granted = true;
40      }
41
42      function vote(string calldata ballot) external _notEndend {
43          // Ya se ha emitido un voto con esta cuenta
44          require(abi.encodePacked(votes_[msg.sender].ballot).length <= 0, "This account
            ↪  has already been used to vote");
45          // Ya se ha emitido un voto con esta cuenta que ha sido borrado
46          require(!votes_[msg.sender].deleted, "This account has been removed");
47          // El usuario está autorizado a votar
48          require(votes_[msg.sender].granted, "This account hasn't been granted to vote");
49
50          votes_[msg.sender].ballot = ballot;
51          votes_address_.push(msg.sender);
52      }
53
54      function revoke(address voter) external _ownerOnly _notEndend {
55          bool deleted = false;
56
57          for (uint i = 0; i < votes_address_.length; i++) {
58              if (votes_address_[i] == voter) {
59                  // Eliminar dirección de el array de votos
60                  votes_address_[i] = votes_address_[votes_address_.length - 1];
61                  votes_address_.pop();
62
63                  // Guardar la dirección
64                  deleted_votes_address_.push(voter);
65                  votes_[voter].deleted = true;
66
67                  deleted = true;
68                  break;
69              }
70          }
71
72          require(deleted == true, "Vote didn't exist");
73      }
74
75      function tally() external _ownerOnly _notEndend returns (string memory) {
76          uint256 result_size = (129 * candidates_count_) + 1;
77          bytes memory result = new bytes(result_size);
78
79          uint256 ballot_size = bytes(votes_[votes_address_[0]].ballot).length;
80
81          bytes memory sum_input = new bytes(ballot_size + result_size + 64);
```

```
82          uint256 sum_input_size = sum_input.length + 32;
83
84          bytes memory public_key = bytes(public_key_);
85          uint256 public_key_size = public_key.length + 32;
86
87          bytes memory verify_input = new bytes(public_key_size + ballot_size + 64);
88          uint256 verify_input_size = verify_input.length + 32;
89
90          uint256 candidates_count = candidates_count_;
91
92          // Crear input pk + n_candidatos + voto
93          for (uint j = 32; j < public_key_size + 32; j += 32) {
94              assembly {
95                  mstore(add(verify_input, j), mload(add(public_key, sub(j, 32))))
96              }
97          }
98
99          assembly {
100             mstore(add(verify_input, add(public_key_size, 32)), candidates_count)
101         }
102
103         for (uint i = 0; i < votes_address_.length; i++) {
104             string memory ballot = votes_[votes_address_[i]].ballot;
105
106             // Añadir voto a verify_input
107             for (uint j = public_key_size + 64; j < public_key_size + ballot_size + 96; j
                ↪  += 32) {
108                 assembly {
109                     mstore(add(verify_input, j), mload(add(ballot, sub(j,
                        ↪  add(public_key_size, 64)))))
110                 }
111             }
112
113             // Crear sum_input voto + result
114             for (uint j = 32; j < ballot_size + 64; j += 32) {
115                 assembly {
116                     mstore(add(sum_input, j), mload(add(ballot, sub(j, 32))))
117                 }
118             }
119             for (uint j = 64 + ballot_size; j < result_size + ballot_size + 96; j += 32)
                ↪  {
120                 assembly {
121                     mstore(add(sum_input, j), mload(add(result, sub(j, add(64,
                        ↪  ballot_size)))))
122                 }
123             }
124
125             uint[1] memory verified;
126
127             assembly {
```

```
                if iszero(staticcall(not(0), 0xc, verify_input, verify_input_size,
                ↪  verified, 32)) {
                    revert(0, 0)
                }

                if eq(mload(verified), 1) {
                    if iszero(staticcall(not(0), 0xb, sum_input, sum_input_size,
                    ↪  add(result, 32), result_size)) {
                        revert(0, 0)
                    }
                }
            }
        }

        ended_ = true;
        result_ = result;
        return string(result);
    }
}

```

A.2. Modificaciones en el código de Geth

```
type encryptedSum struct {}

func (c *encryptedSum) RequiredGas(input []byte) uint64 {
        return uint64(len(input)+31)/32*params.IdentityPerWordGas +
        ↪  params.IdentityBaseGas
}
func (c *encryptedSum) Run(input []byte) ([]byte, error) {
        fmt.Printf("sum: %v\n", input)
        ballot_size := new(big.Int).SetBytes(input[32:64]).Uint64()
        ballot := input[64:ballot_size + 64]

        result_size := new(big.Int).SetBytes(input[ballot_size + 64:ballot_size +
        ↪  96]).Uint64()
        result := input[ballot_size + 96:ballot_size + result_size + 96]

        cmd := exec.Command("/usr/local/bin/precompiled-contracts", "--sum",
        ↪  fmt.Sprintf("%v", string(ballot)), fmt.Sprintf("%v",
        ↪  string(bytes.Trim(result, "\x00"))))

        var out bytes.Buffer
        var stderr bytes.Buffer
        cmd.Stderr = &stderr
        cmd.Stdout = &out

```

```
        err := cmd.Run()
        if err != nil {
                fmt.Print("ERROR: ")
                fmt.Println(fmt.Sprint(err) + ": " + stderr.String())
                return []byte{}, err
        }

        fmt.Println("---------------SUM SUCCESS---------------")
        fmt.Printf("Output size: %v\n", len(out.String()))
        return out.Bytes(), nil
}
```

A.3. Contrato precompilado

```
pub fn sum_ballot_option(ballot: &String, acc: &String) -> Result<(), Box<dyn Error>> {
    let parsed_ballot: EncryptedChoice<Ristretto, SingleChoice> =
    ↪  serde_json::from_str(&ballot)?;

    let mut parsed_acc: Vec<Ciphertext<Ristretto>> = match serde_json::from_str(&acc) {
        Ok(res) => res,
        Err(_) => vec![Ciphertext::<Ristretto>::zero();
        ↪  parsed_ballot.choices_unchecked().len()],
    };

    assert_eq!(parsed_ballot.len(), parsed_acc.len());

    for (i, choice) in parsed_ballot.choices_unchecked().iter().enumerate() {
        parsed_acc[i] += *choice;
    }

    let parsed_result = serde_json::to_string(&parsed_acc)?;

    print!("{parsed_result}");

    Ok(())
}

pub fn verify_vote(
    pub_key_bytes: &Vec<u8>,
    vote: &String,
    options_count: usize,
) -> Result<(), Box<dyn Error>> {
    let receiver = PublicKey::<Ristretto>::from_bytes(&pub_key_bytes)?;
    let params = ChoiceParams::single(receiver, options_count);
    let ballot: EncryptedChoice<Ristretto, SingleChoice> = serde_json::from_str(&vote)?;

    ballot.verify(&params)?;
```

```rust

    Ok(())
}

fn main() -> Result<(), Box<dyn Error>> {
    let args = env::args().collect::<Vec<String>>();

    eprintln!("{:?}", args);

    match args[1].as_str() {
        "--sum" => sum_ballot_option(&args[2], &args[3]),
        "--verify" => {
            let bytes = BASE64_STANDARD.decode(args[2].clone())?;

            verify_vote(&bytes, &args[3], args[4].parse::<usize>().unwrap())
        }
        _ => panic!("Unknown parameter"),
    }
}
```

Bibliografía

[1] La compañía de elecciones - Smartmatic, https://www.smartmatic.com/es/, accedido el 23-05-2024.

[2] Hsm, https://es.wikipedia.org/wiki/HSM, accedido el 23-05-2024.

[3] S. Park, M. Specter, N.Ñarula, R. L. Rivest, Going from bad to worse: from internet voting to blockchain voting, Journal of Cybersecurity 7 (1) (2021) tyaa025.

[4] R. Taş, Ö. Ö. Tanrıöver, A systematic review of challenges and opportunities of blockchain for e-voting, Symmetry 12 (8) (2020) 1328.

[5] U. Jafar, M. J. A. Aziz, Z. Shukur, Blockchain for electronic voting system—review and open research challenges, Sensors 21 (17) (2021) 5874.

[6] L. Hurtado Coca, Análisis de la seguridad en la propuesta de voto electrónico con blockchain, Ph.D. thesis (2022).

[7] F. Þ. Hjálmarsson, G. K. Hreiðarsson, M. Hamdaqa, G. Hjálmtỳsson, Blockchain-based e-voting system, in: 2018 IEEE 11th international conference on cloud computing (CLOUD), IEEE, 2018, pp. 983–986.

[8] F. D. Giraldo, C. E. Gamboa, et al., Electronic voting using blockchain and smart contracts: Proof of concept, IEEE Latin America Transactions 18 (10) (2020) 1743–1751.

[9] C. A. Rodríguez Burgos, H. O. Franco Másmela, et al., Prototipo de votación electrónica basada en blockchain, caso de estudio: Procesos electorales en la universidad piloto de colombia (2020).

[10] W. Qu, L. Wu, W. Wang, Z. Liu, H. Wang, A electronic voting protocol based on blockchain and homomorphic signcryption, Concurrency and Computation: Practice and Experience 34 (16) (2022) e5817.

[11] J. Chandra Priya, P. R. Sathia Bhama, S. Swarnalaxmi, A. Aisathul Safa, I. Elakkiya, Blockchain centered homomorphic encryption: A secure solution for e-balloting, in: Proceeding of the International Conference on Computer Networks, Big Data and IoT (ICCBI-2018), Springer, 2020, pp. 811–819.

[12] H. Kim, K. E. Kim, S. Park, J. Sohn, E-voting system using homomorphic encryption and blockchain technology to encrypt voter data, arXiv preprint arXiv:2111.05096 (2021).

[13] Cadena de bloques, https://es.wikipedia.org/wiki/Cadena_de_bloques, accedido el 21-05-2024.

[14] Aritmética modular, https://es.wikipedia.org/wiki/Aritm%C3%A9tica_modular, accedido el 21-05-2024.

[15] TypeScript, https://www.typescriptlang.org/, accedido el 21-05-2024.

[16] React, https://react.dev/reference/react, accedido el 21-05-2024.

[17] HTML, , accedido el 21-05-2024.

[18] SEO, https://developers.google.com/search/docs/fundamentals/seo-starter-guide?hl=es, accedido el 22-05-2024.

[19] HTTP, https://developer.mozilla.org/es/docs/Web/HTTP, accedido el 21-05-2024.

[20] Ethereum Yellow Paper, https://ethereum.github.io/yellowpaper/paper.pdf, accedido el 21-05-2024.

[21] Bitcoin, https://bitcoin.org/es/como-funciona, accedido el 24-05-2024.

[22] ECDSA, https://es.wikipedia.org/wiki/ECDSA, accedido el 23-05-2024.

[23] JavaScript, https://developer.mozilla.org/es/docs/Web/JavaScript, accedido el 21-05-2024.

[24] Turing completo, https://es.wikipedia.org/wiki/Turing_completo, accedido el 21-05-2024.

[25] A. P. Gregory Maxwell, Borromean Ring Signatures (2015).

[26] Transformada de Fiat-Shamir, https://en.wikipedia.org/wiki/Fiat%E2%80%93Shamir_heuristic, accedido el 21-05-2024.

[27] Rust, https://www.rust-lang.org/es, accedido el 21-05-2024.

[28] elastic-elgamal - Rust, https://docs.rs/elastic-elgamal/latest/elastic_elgamal/, accedido el 21-05-2024.

[29] blind-rsa-signatures - Rust, https://docs.rs/blind-rsa-signatures/latest/blind_rsa_signatures/, accedido el 21-05-2024.

[30] ecies - Rust, https://docs.rs/ecies/latest/ecies/, accedido el 21-05-2024.

[31] RSA - Rust, https://docs.rs/rsa/latest/rsa/, accedido el 21-05-2024.

[32] WebAssembly, https://webassembly.org/, accedido el 21-05-2024.

[33] wasm-pack, https://rustwasm.github.io/docs/wasm-pack/, accedido el 23-05-2024.

[34] Geth, https://geth.ethereum.org/docs/getting-started, accedido el 21-05-2024.

[35] FFI, https://doc.rust-lang.org/nomicon/ffi.html, accedido el 21-05-2024.

[36] PostgreSQL, https://www.postgresql.org/, accedido el 21-05-2024.

[37] shadcn/ui, https://ui.shadcn.com/, accedido el 23-05-2024.

[38] SSL, https://es.wikipedia.org/wiki/Seguridad_de_la_capa_de_transporte, accedido el 21-05-2024.

[39] Nginx, https://nginx.org/en/, accedido el 21-05-2024.

[40] PKCS12, https://en.wikipedia.org/wiki/PKCS_12, accedido el 24-05-2024.

[41] PKCS1, https://en.wikipedia.org/wiki/PKCS_1, accedido el 24-05-2024.

[42] React context, https://react.dev/reference/react/useContext, accedido el 21-05-2024.

[43] Go, https://go.dev/, accedido el 21-05-2024.

[44] A. Amores Martínez, Blockchain, algoritmos de consenso (2020).

[45] Bifurcación dura, https://es.wikipedia.org/wiki/Bifurcaci%C3%B3n_dura, accedido el 21-05-2024.

[46] SMTP, https://www.ibm.com/docs/es/i/7.3?topic=information-smtp, accedido el 22-05-2024.

[47] Amazon EC2, https://aws.amazon.com/es/ec2/, accedido el 23-05-2024.

[48] Cloud Computing - Servicios de informática en la nube - AWS, https://aws.amazon.com/es/, accedido el 23-05-2024.