
Universidad Michoacana de San Nicolás de Hidalgo

Facultad de Ciencias Físico Matemáticas

# Sistema de votaciones electrónicas bajo un esquema de blockchain

**T E S I S**

que para obtener el título de:

licenciado en ciencias físico matemáticas

P R E S E N T A :

Marco Antonio Nava Aguilar

tutora

Dra. Karina Mariela Figueroa Mora

Ciudad Universitaria, Morelia, Michoacán. Febrero 2021

# Resumen

Debido a la situación sociopolítica en la que se encuentra el país y las diversas cuestiones colaterales a ésta que afectan al desarrollo de los comicios, en México los procesos electorales siempre son motivo de controversia. Por esta razón, se busca proponer una solución empleando la estructura de datos *Blockchain*. El objetivo es que se logre desarrollar un sistema de votación electrónica que sea inquebrantable y confidencial respecto a quién emite los votos, y que mantenga también una condición de transparencia en cuanto a la cantidad y contenido de las participaciones. Cabe mencionar que se generaría una base de datos pública y anónima que sirva de ayuda para revisar y verificar el conteo de votos, eliminando así el error humano posible en la manifestación análoga de la democracia, así como abrir la posibilidad de que un ciudadano pueda comprobar que su voto fue respetado y tomado en cuenta, haciendo uso de su llave privada. Además, con esta solución será posible reducir gastos en materiales físicos desechables que se usan en las votaciones tradicionales, por supuesto, además se reduciría la contaminación.

**Conceptos clave:** criptografía de llave pública, voto por internet, función hash, blockchain, criptografía de firma ciega.

# Abstract

Due to the socio-political situation of the country, and its collateral issues that affect the development of the elections, the electoral processes are always controversial. On that score, we propose a solution of electronic voting employing the data structure used in criptocurrencies, the *Blockchain*. The main objetive to achieve, is to develop an electronic voting system that preserves the properties of security and anonimity of the users, and also being transparent in respect of the number and content of the ballots. With that, a public database is generated that can be used by the citizens to see the votes and verify the vote count, removing the posible human error; also, one voter can make sure that his vote has been respected using his unique private key. Also, this solution is meant to reduce the ecological damage caused by the use of disposable physical materials used by conventional elections.

# Índice general

# Lista de Algoritmos

# Capítulo 1

# Introducción

Para un sistema democrático, el ejercicio de la expresión de la opinión pública libre y efectiva es fundamental para emitir no solo las necesidades de aquellos que lo conforman, sino también sirve como diagnóstico de aquello que acontece en su interior, sus problemas y sus posibles soluciones de acuerdo a lo que el pueblo demanda. De esta manera una democracia saludable es aquella en la que las partes que la conforman pueden emitir su voz, y la voz de la mayoría es capaz de causar cambios efectivos en el sistema.

En la mayoría de países existentes, la forma de gobierno predominante es alguna forma de democracia en la que los ciudadanos eligen a sus gobernantes por medio de votaciones. Por la naturaleza de esta misma actividad, su magnitud es a gran escala y en algunas regiones puede llegar a ser masiva en aquellos lugares donde su densidad de población es elevada, de modo que en pleno siglo XXI, mientras el planeta afronta grandes problemas de contaminación global por productos desechables, el ejercicio tradicional de la democracia por medio de medios físicos de papel y lápiz resulta difícil de sostener sin repercusiones de carácter sustentables o económicas, pues el costo de una elección de este tipo es elevado.
Por otro lado, podemos observar también en la importancia de la democracia que es un método en el que una persona es elegida para llegar al poder de una nación y a pesar de que existen métodos para hacer efectiva la validez de las elecciones, existen

también regiones del planeta en la que se busca vulnerar esta efectividad para conseguir que alguna persona llegue al liderazgo de la nación sin pasar por el consenso popular, mediante algún tipo de fraude electoral.

Tomando como ejemplo el proceso electoral en México, este consiste en 3 pasos:

1. Escoger a los representantes de las casillas electorales. Las autoridades escogen ciudadanos al azar para ser parte del personal de las casillas electorales el día de la elección, éstos serán entrenados para el proceso. Sus funciones principales serán otorgar las boletas electorales a los ciudadanos que acudan a votar, registrar la asistencia del ciudadano a la casilla, y cuidar que todo fluya de forma correcta de modo que se cumpla el ejercicio de la democracia

2. Durante el día de las votaciones, después de que los representantes validan la existencia del ciudadano en el padrón electoral y otorgan la papeleta de los candidatos a elegir, el ciudadano procede a resguardarse en una cabina donde marcará la boleta con su elección, y posteriormente depositará su participación en la casilla destinada a esto.

3. Una vez finalizada la jornada electoral, se procede a realizar el conteo a mano de los votos por parte de los representantes de casilla, generar un reporte de resultados, y colgar dicho reporte al exterior del lugar donde la votación fue llevada a cabo. Los resultados son llevados también a una junta distrital correspondiente del Instituto encargado de llevar a cabo las votaciones.

Como podemos observar, este proceso es vulnerable en su integridad ya en procesos electorales se ha reportado robo y destrucción de boletas, cancelaciones y reinstalaciones de casillas, compra de votos y errores en el conteo final provocados por factores humanos.

Más aún, a finales del año 2019, una nueva emergencia sanitaria surge en la región de Wuhan en China originada a causa de una nueva cepa de coronavirus (el virus

SARS-CoV-2). Este nuevo virus se extendió de manera exponencial debido a su efectividad de contagio y a sus mecanismos de transmisión a través de las secreciones respiratorias de las personas infectadas, sobre todo mediante la expulsión, a través de la tos o el estornudo, de pequeñas gotas y aerosoles que pueden cruzar el aire, además de contacto directo con estas secreciones o por objetos contaminados por las mismas o fómites (también conocidos como vectores pasivos).

La recomendaciones de prevención y medidas de higiene para combatir la pandemia ocasionada por el COVID-19 (nombre con el que se le conoció a la enfermedad ocasionada por el SARS-CoV-2) incluyeron el aislamiento del público en general dentro de sus hogares, el cierre de negocios considerados "no esenciales", la implantación de controles de ingreso y egreso en lugares "esenciales", y la cancelación de eventos de asistencia masiva. La pandemia y los efectos de las medidas mencionadas se extendieron en diferentes medidas durante todo el año 2020 y a lo largo de todo el mundo, interfiriendo incluso en actividades programadas para ese año en distintas regiones del globo. Dado que fueron limitadas las actividades, también se interfirió con los comicios para la elección de máximos órganos de gobierno de entidades como universidades, teniéndose que llevarse éstos a cabo ya sea de manera remota, o bien no realizándose en absoluto.

Debido a lo descrito en las líneas anteriores, y con la finalidad de evitar diversos delitos electorales aplicables a sistemas físicos (como son la coerción mediante la solicitud de un comprobante físico, las cadenas de votos, alteración de urnas y boletas, entre otros.), se han desarrollado y llevado a cabo en diversas partes del mundo sistemas de votaciones innovadores que busquen resolver problemáticas específicas, que puedan servir a un ejercicio de la democracia más efectivo, no carentes también de sus defectos. En el presente trabajo se trabajará una propuesta de un sistema de votaciones electrónicas, por esta misma razón, también se dará revisión a los orígenes de dichos sistemas, los antecedentes y al estado del arte en esta materia.

## 1.1. Objetivo

La presente tesis tendrá como objetivo implementar las bases de un sistema de votación electrónica que mantenga los siguientes principios:

1. Anonimización del votante.
2. Integridad y confiabilidad de cada voto emitido.
3. Transparencia por cada voto emitido.
4. Precisión en el conteo de votos.

## 1.2. Estructura de la tesis

Esta tesis será desarrollada en los siguientes capítulos:

- **Capítulo 1:** Este capítulo contiene la introducción de la tesis, donde se plantea el problema que se quiere solucionar con el trabajo, y los objetivos que se piensan cumplir. También describe la estructura del trabajo.
- **Capítulo 2:** En este capítulo se abordan de manera breve los antecedentes de los sistemas electrónicos, cuáles fueron las primeras aproximaciones a un sistema que cumpliera con los objetivos enunciados con anterioridad, cuáles son sus clasificaciones, qué ha fallado, qué ha funcionado, y como se puede usar lo ya existente en el desarrollo de esta propuesta.
- **Capítulo 3:** Antes de exponer a detalle la propuesta desarrollada, es necesario desarrollar la teoría en la que se sustenta, pues es a partir de esta de donde heredará las propiedades que se desean que el sistema cumpla. Esa será la finalidad de este capítulo.
- **Capítulo 4:** En este capítulo se da a conocer el sistema desarrollado, se extenderá la teoría relacionada a los objetivos a cumplir, cómo se le da uso a

la teoría mencionada en el capítulo 3 y cuáles son los pasos a seguir para su correcto funcionamiento al desglosarlo por etapas y algoritmos.

- **Capítulo 5:** Este capítulo habla sobre las conclusiones sobre el presente trabajo, si se cumplen los objetivos y en caso afirmativo, cómo es que se logran. También se trata aquello que necesita correcciones para evitar los fallos y como se plantea corregirlo en el futuro.

- **Bibliografía:** Al final del documento se encuentran las referencias utilizadas en el desarrollo capítulos enumerados anteriormente. Estas referencias son listadas en orden alfabético teniendo en cuenta la primera letra del apellido del primer autor (o en su defecto del único autor).

# Capítulo 2

# Estado del arte

Dada la importancia del proceso electoral, para poder discutir la propuesta de un nuevo sistema es necesario entender lo que se requiere para que un sistema de votaciones sea considerado tal, y en consecuencia, los riesgos que podría significar realizar algún cambio en los procesos existentes, de modo que se pueda garantizar una transición segura y eficiente de un sistema análogo a uno electrónico.

En un sistema de votaciones son considerados tres aspectos de verificabilidad [29][21]: individual, universal y elegibilidad. La *verificabilidad individual* permite al votante comprobar que su voto es incluido en el resultado electoral. La *verificabilidad universal* permite al votante o a los observadores electorales comprobar que el resultado electoral corresponde a los votos emitidos. Y la *elegibilidad* permite a los votantes y observadores comprobar que cada voto en el resultado electoral fue emitido por un votante único y registrado.

En la literatura es bien conocido el hecho de la desconfianza existente en las personas, respecto al código que no ha sido totalmente escrito por ellos mismos [31], no obstante, el concepto de la verificabilidad de las elecciones a través de soluciones criptográficas ha emergido en la literatura académica para introducir transparencia y confianza dentro de los sistemas de votación [6][18]. Éstas soluciones permiten a los votantes y a los observadores electorales generar confianza en los sistemas debido

a que les permite verificar que los votos han sido registrados y declarados correctamente, de manera independiente al hardware y al software sobre el que se ejecuta la elección.

En este capítulo se buscará explorar el camino recorrido en la búsqueda de lograr ésta transición en la expresión de la democracia.

## 2.1. Antecedentes

La intención de revolucionar la forma de almacenar y procesar información otorgada mediante consulta popular se remonta al siglo XIX. Tenemos en principio la patente obtenida en 1889 por el Dr. Herman Hollerith, para su *Máquina tabuladora eléctrica*, la cual fue usada en el censo de 1890. Ésta funcionaba mediante la ayuda de perforaciones en tarjetas de $3{,}25 \times 6{,}625$ pulgadas con 12 filas y 24 columnas de agujeros circulares como la del inciso *a*) la figura 2.1 [17]. Dichas tarjetas eran procesadas en el tabulador, que usaba solenoides para incrementar contadores mecánicos dentro del artefacto. Al introducir la tarjeta en el lector, un conjunto de cables con resorte suspendidos dentro de éste hacían presión sobre la primera, que quedaba sumergida sobre frascos llenos con mercurio, como en el inciso b) de la figura 2.1. Debido a que el mercurio se situaba debajo de las posibles perforaciones en la carta, éstas permitían que los cables entraran en contacto con el mercurio, creando un enlace eléctrico [9] que podía usarse para contar, clasificar y hacer sonar una campana para que el operador sepa que la tarjeta había sido leída. El tabulador tenía 40 contadores, cada uno con un cuadrante dividido en 100 divisiones, con dos agujas indicadoras; uno que avanzaba una unidad con cada pulso de conteo, el otro que avanzaba una unidad cada vez que el otro dial daba una revolución completa. Este arreglo permitió un recuento de hasta 9,999.

Por otro lado, la idea de votar con solo presionar un botón, usando tecnología electrónica la podemos encontrar en 1898, cuando Frank S. Wood de Boston registró

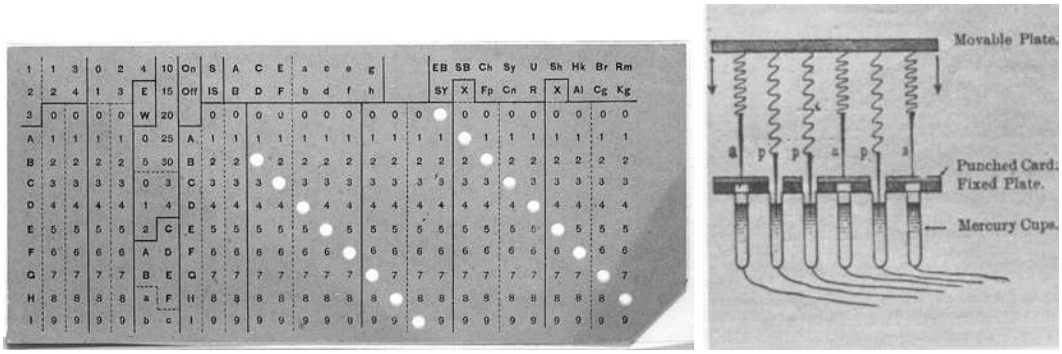



Figura 2.1: a)Tarjeta perforada; b)Lector de tarjetas del tabulador

una patente sobre una máquina de voto electrónico mediante el registro directo [33]. En años anteriores, otros hicieron sus respectivos intentos con casillas electorales que funcionaban mediante palancas. Tal es el caso de Thomas Alva Edison, cuya patente del 1 de junio de 1869 fue pensada para contabilizar los votos nominales de las cámaras legislativas [11].

Como podemos encontrar en la patente de Frank S. Wood (s. XIX) [33], su invención fue originada con los siguientes objetivos:

1. Acabar con las perplejidades de las boletas impresas, así como con los errores originados del conteo subsecuente.

2. Otorgar al votante secrecía absoluta y el medio más simple para registrar su elección de candidatos con la máxima expedición.

3. Proporcionar los medios por los cuales el votante no solo puede registrar su elección de candidatos, sino también computar su voto con todos los demás previamente registrados por medios invisibles automáticos.

4. Tener todas las garantías necesarias, no solo para proteger al votante en su privilegio legal, sino también para prevenir el fraude, de modo que ningún votante pueda votar más de una vez por el mismo candidato ni por más de un candidato para el mismo cargo

5. Proporcionar los medios por los cuales, cuando se cierren las urnas, los oficiales

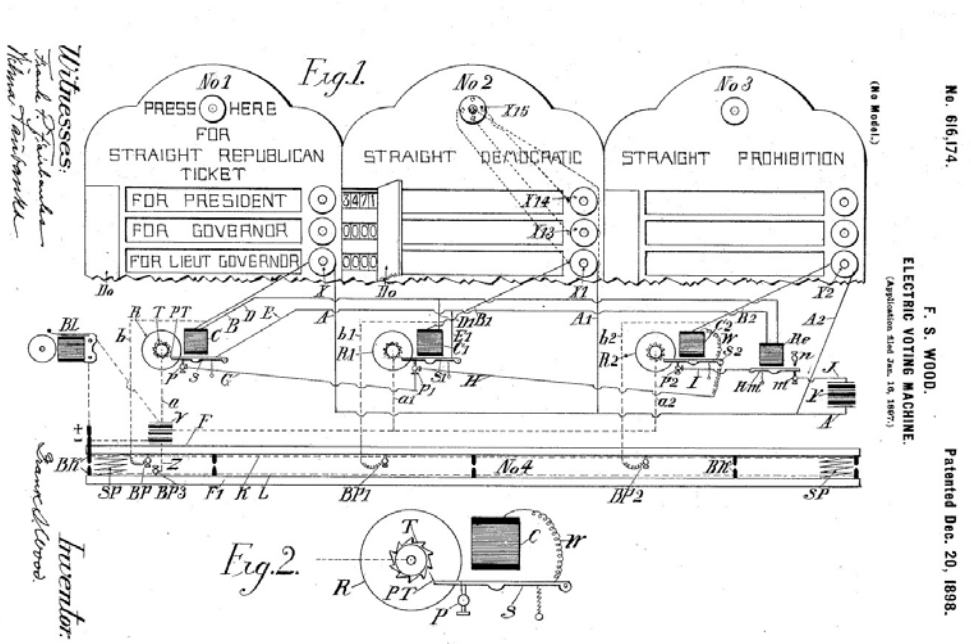



Figura 2.2: Imagen de la patente de Frank S. Wood

electorales correspondientes puedan tener acceso al voto total calculado para todos los candidatos, que, al ser transcrito y certificado, será el voto oficial exacto.

Por otro lado, como se puede ver en la figura 2.2 el dispositivo era bastante simple y funcionaba mediante botones. Un botón en la parte superior permitía cambiar entre partidos electorales, mientras que en el cuerpo de la casilla se contaba con un botón para cada candidato. Estos, al ser presionados activaban circuitos eléctricos que registraban el voto.

Éstas invenciones sentaron los precedentes sobre los cuales, con el tiempo, se fueron desarrollando los sistemas electrónicos de votación que tenemos hoy en día. Sin embargo, ¿Cuáles son estos sistemas? ¿En qué consisten? ¿Cuáles son sus ventajas y desventajas?

## 2.2. Sistemas de Votación Electrónica actuales

El objetivo de esta sección es mostrar y analizar los sistemas de votaciones electrónicas existentes y que han sido aplicados en la vida real; además desglosar y diferenciar las ventajas y desventajas de cada uno. Esto con la finalidad de vislumbrar los fallos que han hecho que no se hayan implementado con generalidad en todas las democracias, y así poder otorgar una propuesta que sea capaz de subsanar estos errores y desventajas.

### 2.2.1. Votación Electrónica de Registro Directo (DRE)

Como fue mencionado en la sección anterior, el primer sistema de votaciones completamente electrónicas en aparecer fue el Sistema de Votación Electrónica de Registro Directo (o Direct-Recording Electronic por sus siglas en inglés), proveniente de la invención de Frank S. Wood. En la actualidad, en este sistema el ciudadano emite su voto en una papeleta generada en una pantalla que es manipulada por botones, cursores, o bien una pantalla táctil, y su opinión queda registrada dentro de la misma máquina. Los votos quedan registrados ya sea en una memoria local, física, o bien ser transferidos por medio de una red como Internet o de telefonía pública a una oficina central, a estas últimas se les conoce como DRE de red pública.

Entre sus principales ventajas es el no requerir de impresión de boletas, lo cual reduce el impacto económico que conlleva, así como también el gasto ecológico que generan una vez que son desechadas, además de eliminar el problema del posible desfase existente al no coincidir la cantidad de boletas impresas con la de asistentes a las casillas de votación. Por otro lado, al no haber necesidad de personas encargadas del conteo final de los votos, se elimina el error humano que los observadores electorales que realizan esta tarea puedan cometer. Este sistema está pensado para ser un sistema amigable con el votante, por lo que su uso es muy sencillo y rápido, permitiendo que incluso personas con capacidades diferentes ejerzan su derecho.

Sin embargo, de lado de las desventajas, se tiene que al carecer de un registro tangible, se elimina la seguridad de haber efectuado una acción indeleble, dado que un registro electrónico puede ser alterado [32], además, este sistema puede ser intervenido fácilmente (tanto en hardware como en software) por personas que tengan acceso físico a la máquina, afectando a las tarjetas de memoria y los registros contenidos en ellas [14]. Y hablando específicamente de los DRE de red pública, estos son vulnerables a los ataques *Man in the middle*, que consiste en que el atacante se sitúa secretamente entre dos partes que intentan comunicarse y adquiere la capacidad de leer y posiblemente modificar la información intercambiada entre las partes mientras que estas confían en que se comunican directamente entre sí. Estos ataques podrían hacer perder los votos emitidos o bien, que el contenido de los mismos sea modificado.

### 2.2.2. Votación Electrónica en Papel

El siguiente sistema existente a evaluar es la Votación Electrónica en Papel, que viene directamente de la invención de Herman Hollerith. Este sistema es muy parecido al sistema tradicional que conocemos, utiliza boletas impresas en papel que el ciudadano emite su opinión mediante perforaciones, marcas especiales (i.e. rellenar círculos impresos en boletas), y más recientemente lápices digitales. Después de esto, las boletas son digitalizadas para ser contadas mediante una máquina, además cada boleta es contada de forma manual. Estos sistemas están diseñados mediante técnicas de criptografía para que sea preservada la privacidad, la precisión y la verificabilidad de los votos [28].

Al ser tan parecido al sistema de votación que se usa normalmente, no es necesario capacitar a la gente para el uso de este otro sistema. En caso de que haya desconfianza en el conteo ejercido por la máquina, existen los comprobantes físicos, los cuales pueden ser contados manualmente. Al tratarse de máquinas, personas con diferentes capacidades pueden participar con su sufragio debido a los sistemas de accesibilidad integrados. Y por último, al ser al mismo tiempo un sistema análogo y electrónico, independientemente de cuales sean las fallas del software, se pueden usar las papeletas

de la forma convencional.

Desafortunadamente, también esto último representa una desventaja, pues aunque la parte electrónica esté completamente funcional, su virtud análoga se convierte en vicio al ser vulnerable a todos los métodos de fraude convencionales (robo de boletas, compra de votos, cadenas de votación, etc.). Además, igual que el DRE, el hardware es vulnerable a diversas técnicas de manipulación que llevan a un resultado alejado de la realidad. Asimismo, los sistemas basados en criptografía son fuertes debido a principios matemáticos; no obstante en muchos casos la teoría detrás de estos es superficial, débil, y en otros casos insuficiente para garantizar una votación efectiva y segura, además del hecho mencionado con anterioridad, en el que la gente puede desconfiar del código que no ha sido programado por ellos [10][31].

### 2.2.3. Voto por Internet

El más reciente, y el último de los sistemas existentes a analizar es la Votación por Internet. Su principal característica es el uso de redes de telecomunicaciones para depositar el voto del usuario en urnas con una ubicación distinta a la del mismo usuario de manera instantánea, permitiendo así que un ciudadano lejos del lugar de votación pueda ejercer su derecho. A pesar de que puede ser sincronizado con los sistemas anteriores, su flexibilidad le permite que su implementación también sea simple y sin necesidad de hardware costoso y sofisticado.

Entre sus ventajas se encuentran que dada la naturaleza de este sistema, es posible acceder a las urnas virtuales desde cualquier parte del mundo siempre que el votante esté conectado a Internet. Por otro lado, debido que no son usados componentes físicos para este sistema, la relación coste-voto es eficiente, sin embargo, debido a esto no existe una muestra de respaldo tangible, lo que genera desconfianzas en el votante. Además, estos sistemas centrados específicamente en el software tienen puntos destacables.

Respecto a *front-end* (la parte del lado del usuario en una aplicación que es la que interactúa con él), la accesibilidad económica permite los desarrolladores crear una interfaz de usuario que sea diseñada de manera intuitiva y de fácil uso para cualquier votante, que además tenga características de accesibilidad que darían inclusión a todo tipo de usuarios. No obstante, una que fue diseñada de manera deficiente generaría más problemas de los que daría solución. Por el lado del *back-end* (la parte de interior de una aplicación encargada de procesar la información del usuario del lado del servidor), este sistema es vulnerable a *hackers* y ataques de diversos tipos, como la versión en Internet de *Man in the middle*, el *phishing* en el que atacantes se hacen pasar por alguna entidad oficial con la finalidad de obtener datos confidenciales del votante, ataques DDoS (Denegación Distribuida de Servicio por sus siglas en inglés) que consiste en enviar varias solicitudes al recurso web atacado, con la intención de desbordar la capacidad del sitio web para administrar varias solicitudes y de evitar que este funcione correctamente, entre otros; además de existir fallas en la identificación, dado que puede llevarse a cabo fácilmente una suplantación de identidad, ataque en el que una tercer persona (conocida por el votante o no) se hace pasar por el votante, con lo que no garantiza que vota la persona que dice ser.

# Capítulo 3

# Marco Teórico

Una vez visto el trayecto que se ha recorrido en este tema, es necesario que se desarrollen ciertos conceptos teóricos en los que se asienta la propuesta que se presenta en este trabajo. El objetivo de este capítulo es presentar dichas bases teóricas, de modo que sea una explicación sencilla y completa, sin llegar a ser demasiado extensiva.

En primer lugar se desarrolla la información de las funciones criptográficas que sirven como corazón de la propuesta, estas funciones son las funciones hash criptográficas (concretamente las funciones SHA-1, SHA-256 y HMAC), el Algoritmo de Firma Digital de Curva Elíptica, un Algoritmo de Firma Digital Ciega, y un Algoritmo asimétrico (es decir, un algoritmo de encriptación de llave pública). Se sigue con el desarrollo de información acerca de los sistemas biométricos (específicamente los basados en huella digital) cuyas bases servirán para la seguridad en la identificación del usuario. Una vez terminado esto, se continua con la teoría de *Blockchain*, que es la estructura de datos que sirve como esqueleto de la propuesta a desarrollar, esto incluye las estructuras y algoritmos que permiten su correcto funcionamiento. Finalmente, se concluye con las definiciones de términos técnicos de computación que se usarán en este trabajo.

## 3.1. Funciones Hash

Una **función hash** $H$ es una transformación que toma una entrada $m$ y regresa una cadena de caracteres de tamaño fijo, que es llamado el valor hash $h$. Un algoritmo hash criptográfico (también llamado "de un solo sentido") debe cumplir 5 requerimientos:

- Resistencia en un solo sentido: para cualquier valor hash $h$, es computacionalmente inviable encontrar $x$ tal que $H(x) = h$
- Determinista: Una función hash se dice que es determinista cuando dada una cadena de entrada siempre devuelve el mismo valor hash. Es decir, el valor hash es el resultado de aplicar un algoritmo que opera solo sobre la cadena de entrada.
- Bajo costo: Encontrar el valor hash $h$ necesita ser económica, ya sea computacionalmente, espacialmente, temporalmente, etc.
- Resistencia a colisiones, de la cual se destacan dos tipos:
  - Resistencia débil: dado cualquier entrada $x$, es computacionalmente inviable encontrar $H(y) = H(x)$, tal que $y \neq x$.
  - Resistencia fuerte: es computacionalmente inviable encontrar $(x, x')$ de modo que $H(x) = H(x')$.
- Efecto avalancha: Se dice que una función hash cumple esta propiedad, cuando un cambio pequeño en una entrada $x$, como un bit, un número, o un carácter, crean cambios enormes en el valor hash $h$. Podemos pensar en esta propiedad como contraria a la continuidad, en la que cambios pequeños en la entrada ocasionan cambios pequeños en la salida.

Estas propiedades las hacen adecuadas para su uso como un medio para comprobar la integridad de un mensaje y como parte de esquemas de firma digital. En este trabajo desarrollaremos la teoría sobre dos funciones bastante difundidas y usadas tanto en individual como en conjunto, que serán también las utilizadas en el capítulo 4, el SHA-1 y el SHA-256.

### 3.1.1. SHA-1 y SHA-256

La familia *Secure Hash Algorith*, mejor conocida como las funciones hash SHA, es una familia de funciones hash criptográficas desarrolladas por el Instituto Nacional de Estándares y Tecnología (NIST en inglés) en asociación con la Agencia de Seguridad Nacional (NSA en inglés) del gobierno de los Estados Unidos de América, con motivo de ser integrada como parte del Digital Signature Standard (DSS), un esquema de firmas digitales que requería de una función hash para su funcionamiento.[25]

En cada uno de estos algoritmos podemos destacar 2 fases: la primera expansión del mensaje seguida de una transformación de actualización de estado que es iterada por un cierto número de rondas, distinto para cada algoritmo de la familia. En las siguientes secciones hacemos uso de los siguientes operadores: operador de desplazamiento de bits (*shift*) a la izquierda ($\ll$) y desplazamiento a la derecha ($\gg$), además de los operadores de rotación a la izquierda ($\lll$) y a la derecha ($\ggg$).

En este apartado se describirán el SHA-1 y el SHA-2, el segundo se refiere al conjunto de funciones hash de dicha familia, el cual está conformado por las funciones SHA-224, SHA-256, SHA-384 y SHA-512. Sus sufijos provienen de la longitud de bits del mensaje que producen al recibir una entrada con tamaño máximo de $2^{64} - 1$ bits para el SHA-1, SHA-224 y SHA-256 y un máximo de $2^{128} - 1$ bits para los demás.

El SHA-1 produce una salida de tamaño 160 bits, la entrada es procesada en partes (o bloques) de 512 bits cada una y es iterada en 84 rondas. El mensaje se empaqueta agregando un uno (“1”) al inicio seguido de ceros (“0”) hasta el bit 448, finalmente la longitud del mensaje es insertada en los 64 bits restantes. El uno inicial es agregado con la finalidad de evitar colisiones. El algoritmo del SHA-1 está dado por el Algoritmo 3.1.1.

**Algoritmo 3.1.1** Algoritmo del SHA-1

**Función** SHA-1(M): ▷ Donde M es el mensaje de entrada
  **Para cada** b en M **hacer**: ▷ b es cada bloque de 512 bits de M
    $W = f_{exp}(B)$;
    $a = H_0; b = H_1; c = H_2; d = H_3; e = H_4$;
    **Para** $i \in \{0, \ldots, 79\}$ **hacer**:
      **Si** $0 \leq i \leq 19$ **entonces**
        $T = a \ggg 5 + f_{if}(b, c, d) + e + W_i + K_0$;
      **si no, Si** $20 \leq i \leq 39$ **entonces**:
        $T = a \ggg 5 + f_{xor}(b, c, d) + e + W_i + K_1$;
      **si no, Si** $40 \leq i \leq 59$ **entonces**:
        $T = a \ggg 5 + f_{maj}(b, c, d) + e + W_i + K_2$;
      **si no, Si** $60 \leq i \leq 79$ **entonces**:
        $T = a \ggg 5 + f_{xor}(b, c, d) + e + W_i + K_3$;
      $e = d; d = c; c = b \ggg 30; b = a; a = T$;
    $H_0 = a + H_0; H_1 = b + H_1; H_2 = c + H_2; H_3 = d + H_3; H_4 = e + H_4$;
  **regresa** concatenación$(H_0 + H_1 + H_2 + H_3 + H_4;)$

Los resultados intermedios de cada bloque son almacenados en cinco bloques de 32 bits denotados $h_0, ..., h_4$. Los cuales son inicializados como sigue:

$$H_0 = 0x67452301,\ H_1 = 0xefcdab89,\ H_2 = 0x98badcfe,\ H_3 = 0x10325476,$$
$$H_4 = 0xc3d2e1f0$$

Además de 4 constantes auxiliares $K_0, \ldots, K_{63}$ con los valores:

$$K_0 = 0x5a827999,\ K_1 = 0x6ed9eba1,\ K_2 = 0x8f1bbcdc,\ K_3 = 0xca62c1d6$$

Donde $f_{exp}$ expande la entrada inicial de 512 bits, consistente de 16 bloques $M_i$ de 32 bits con $i \in \{0, ..., 15\}$ a 80 palabras $W_i$ de 32 bits, con $i \in \{0, ..., 79\}$:

$$W_i = \begin{cases} M_i, & 0 \leq i \leq 15 \\ W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} \ggg 1, & 16 \leq i \leq 79 \end{cases}$$

Por último, definimos también las siguientes funciones:

$$f_{if}(x, y, z) = x \wedge y \oplus \neg x \wedge z,$$
$$f_{maj}(x, y, z) = x \wedge y \oplus x \wedge z \oplus y \wedge z$$
$$f_{xor}(x, y, z) = x \oplus y \oplus z$$

Donde $x, y, z$ son cadenas de 32 bits en dígitos hexadecimales.

A partir de aquí se analizará el SHA-256. Al igual que en el SHA-1, la entrada es procesada en partes (o bloques) de 512 bits cada una y es iterada 84 rondas. El mensaje es empaquetado de la siguiente manera, para que la longitud del mensaje sea múltiplo de 512: Supóngase que $\ell$ es la longitud del mensaje. Primero es añadido un "1" seguido de $k$ dígitos "0", donde $k$ es la mínima solución no negativa a la ecuación $\ell + 1 + k \equiv 448 \ mod \ 512$. Y se agrega al final el bloque de 64 bits equivalente al mensaje. Es agregado el "1" inicial con la finalidad de evitar colisiones. Así el SHA-256 está dado por el algoritmo siguiente:

**Algoritmo 3.1.2** Algoritmo del SHA-256

**Función** SHA256(M): ▷ Donde M es el mensaje de entrada
  **Para cada** b en M **hacer**: ▷ b es cada bloque de 512 bits de M
    $W = f_{exp}(B)$;
    $a = H_0; b = H_1; c = H_2; d = H_3; e = H_4; f = H_5; g = H_6; h = H_7$;
    **Para** $i \in \{0, \ldots, 63\}$ **hacer**:
      $T_1 = h + \Sigma_1(e) + f_{if}(e, f, g) + K_i + W_i$;
      $T_2 = \Sigma_0(a) + f_{maj}(a, b, c)$;
      $h = g; g = f; f = e; e = d + T_1; d = c; c = b; b = a; a = T_1 + T_2$;
    $H_0 = a + H_0; H_1 = b + H_1; H_2 = c + H_2; H_3 = d + H_3$;
    $H_4 = e + H_4; H_5 = e + H_5; H_6 = e + H_6; H_7 = e + H_7$;
  **regresa** concatenación$(H_0 + H_1 + H_2 + H_3 + H_4 + H_5 + H_6 + H_7;)$

Los resultados entre rondas son guardados en 8 registros de 512 bits $H_0$, $H_1$, $H_2$, $H_3$, $H_4$, $H_5$, $H_6$, $H_7$, los cuales son inicializados de la manera siguiente:

$$H_0 = 0x6a09e667 \quad H_1 = 0xbb67ae85 \quad H_2 = 0x3c6ef372 \quad H_3 = 0xa54ff53a$$
$$H_4 = 0x510e527f \quad H_5 = 0x9b05688c \quad H_6 = 0x1f83d9ab \quad H_7 = 0x5be0cd19$$

Además de 64 constantes auxiliares $K_0, \ldots, K_{63}$ que pueden revisarse en [24].
Sea:

$$W_i = \begin{cases} M_i, & 0 \leq i \leq 15 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}, & 16 \leq i \leq 63 \end{cases}$$

Con $M_i$, el bloque $i$ de 512-bits en que se dividió el mensaje y donde $W_i$ es la variable temporal donde se almacena el bloque $M_i$ o bien su modificación. Además de:

$$\Sigma_0(x) = x \ggg 2 \oplus x \ggg 13 \oplus x \ggg 22,$$
$$\Sigma_1(x) = x \ggg 6 \oplus x \ggg 11 \oplus x \ggg 25,$$
$$\sigma_0(x) = x \ggg 7 \oplus x \ggg 18 \oplus x \ggg 3,$$
$$\sigma_1(x) = x \ggg 17 \oplus x \ggg 19 \oplus x \ggg 20,$$

Por último, definimos también las siguientes funciones:

$$f_{if}(x, y, z) = x \wedge y \oplus \neg x \wedge z,$$
$$f_{maj}(x, y, z) = x \wedge y \oplus x \wedge z \oplus y \wedge z$$

Donde $x, y, z$ son cadenas de 32 bits en dígitos hexadecimales.

### 3.1.2. HMAC

En ocasiones es necesario poseer una forma de verificar la integridad de la información transmitida o almacenada en un medio que no es confiable. Los mecanismos que proporcionan dicha verificación de integridad basada en una clave secreta son conocidos como "códigos de autenticación de mensajes" (MAC por sus siglas en inglés). A los MAC basados en una función hash criptográfica se les conoce como HMAC. A los datos se les aplica una función hash de manera iterativa por una compresión en bloques de tamaño $B$. Formalmente [20], es definida como:

$$HMAC(K, m) = H((k \oplus opad) + H(K \oplus ipad + m)) \tag{3.1}$$

Donde:

- $H$ es una función hash criptográfica.
- $k$ es una clave secreta rellena de ceros a la derecha para ser de tamaño $B$ o bien el hash de la llave original si supera dicho tamaño.
- $m$ es el mensaje a ser autenticado.

- $+$ denota concatenación.
- $\oplus$ denota la disyunción exclusiva.
- *opad* es el byte "0x5C" repetido $B$ veces, el relleno exterior.
- *ipad* es el byte "0x36" repetido $B$ veces, el relleno interior.

## 3.2. Firma Digital de Curva Elíptica (FDCE)

La criptografía de curva elíptica es un tipo de criptografía de llave pública basada en el problema del logaritmo discreto (encontrar el valor de $b$, dada la ecuación $a^b = c$, con $a, c$ conocidos) expresado por la suma y multiplicación de puntos de una curva elíptica.

Una curva elíptica $E$ es una curva plana definida por una ecuación de la forma:

$$y^2 = x^3 + ax + b$$

Si $p > 3$ es un número primo, la curva elíptica $E_{\mathbb{Z}_p} : y^2 = x^3 + ax + b$ sobre $\mathbb{Z}_p$ es el conjunto de soluciones $(x, y) \in \mathbb{Z}_p^2$ a la congruencia $y^2 \equiv x^3 + ax + b \bmod p$. $E_{\mathbb{Z}_p}$ forma un grupo con el punto al infinito como identidad [19] y la operación la adición definida por:

$$P + Q = R$$
$$(x_P, x_P) + (x_Q, y_Q) = (x_R, y_R),$$
$$\text{si } \lambda = \tfrac{y_Q - y_P}{x_Q - x_P}$$
$$\text{entonces } x_R = \lambda^2 - x_P - x_Q \ \&\ y_R = \lambda(x_P - x_R) - y_P$$

Con la adición, se define el producto por un escalar $k$, como la adición del punto por sí mismo $k$ veces.

Dado un punto predefinido $g$ generador del grupo y un número aleatorio $k$ se produce un punto en la curva $E_{\mathbb{Z}_p}$ correspondiente a la llave pública $K$ resultante

a operar el punto $g$ y el escalar $k$, el cual es usado como llave privada del usuario. Las firmas digitales son producidas por el conjunto de algoritmos denotados en el Algoritmo 3.2.1 que sigue los siguientes pasos:

1. En primer lugar, el algoritmo "generaLlaveFDCA" un par de llave pública y privada para un usuario, a partir de $G$ generador del grupo y el primo $p$ correspondiente a la curva.

2. El algoritmo "firmado" sigue los siguientes pasos:

   *a*) Se genera un número aleatorio $k$ que será usado como llave privada temporal.

   *b*) A partir de $k$ es generada la llave pública temporal $P$ correspondiente: $P = k * G$, $G$ generador del grupo.

   *c*) Dado que $P$ es un punto en $\mathbb{Z}_p \times \mathbb{Z}_p$, tomamos $R$ la coordenada en el eje $x$ de $P$.

   *d*) Entonces la firma $S$, está dada por $S = k^{-1}(H(m) + Pu * R) \bmod P$, donde $H$ es una función hash y $Pu$ es la llave privada del usuario que firma que es generada en el paso 1, algoritmo "generaLlaveFDCA".

3. La verificación la realiza el algoritmo "Verify" comprobando si $P = S^{-1} * H(m) * G + S^{-1} * R * PPu$ donde $P$ es la llave pública temporal del paso 1, y $PPu$ es la llave pública del usuario que firma.

**Algoritmo 3.2.1** Firma Digital de Curva Elíptica

**Función** GENERALLAVEFDCA$(g, p)$:
    $Pu$; ▷ Número aleatorio $Pu$
    $PPu = Pu * g \bmod p$;
    **regresa** (Pu,PPu);

**Función** FIRMADO$(G, p, Pu, m)$:
    $k$; ▷ Número aleatorio $k$, llave privada temporal
    $P = k * G \bmod p$; ▷ Llave pública temporal
    $R = P[0]$;
    $S = modinv(k, p)(H(m) + Pu * R) \bmod p$;
    **regresa** $S$;

**Función** *Verify*$(P, S, PPu, G, m, p)$:
    $R = P[0]$
    **regresa** $P = modinv(S, p) * H(m) * G + modinv(S, p) * R * PPu$

## 3.3. Esquema de firma digital ciega

En criptografía, un esquema de firma ciega es una tupla de algoritmos de la forma $\mathcal{S} = (\mathcal{S}tp, \mathcal{B}lnd, \mathcal{S}ig, \mathcal{B}lnd^{-1}, \mathcal{V}erify)$, tales que en $\mathcal{S}tp$ se establecen parámetros públicos con los que se generan llaves públicas y privadas para la generación de una firma; mientras que el algoritmo $\mathcal{B}lnd$ será usado para ocultar o cegar el mensaje a firmar, de modo en la que su contenido está oculto a la persona que lo ha firmado. Usando este mensaje cegado y la función $\mathcal{S}ig$ se crea una firma $\sigma$ del mensaje cegado, mismo que podrá ser recuperado con $\mathcal{B}lnd^{-1}$ manteniendo aún la validez de la firma, que puede ser comprobado por cualquier persona con *Verify* usando el mensaje original y $\sigma$. Esto funciona de manera análoga a firmar en el exterior de un sobre sellado en cuyo interior hay un documento con papel pasante y al abrir el sobre, el papel pasante plasmó sobre el documento la firma.

Un esquema de firma digital ciega puede ser pensado como uno que incluye las características de dos sistemas firma digital auténticos combinados de forma especial en un sistema de llave pública conmutativo compuesto por tres funciones [7]:

1. Una función $s$ solo conocida por el firmante, y su correspondiente inversa $s'$ conocida públicamente, de modo tal que $s(s'(x)) = x$ y $s$ no brinda ninguna pista sobre $s'$.

2. Una función conmutativa $c$ y su inversa $c'$, ambas conocidas únicamente por la persona que desea obtener la firma, tales que $c'(s'(c(x))) = s'(x)$ y $c(x)$ con $s'$ no brindan ninguna pista respecto a $x$. A las funciones $c$ y $c'$ se les conoce como funciones de cegado y recuperado respectivamente.

3. Un predicado de redundancia $r$ que comprueba la redundancia suficiente para que la búsqueda de firmas válidas no sea práctica.

Estas funciones otorgan las siguientes propiedades que un esquema de firma digital ciega debe cumplir:

- **Completitud:** Si el firmante y el solicitante de la firma siguen el algoritmo de firma ciega de manera honesta, la verificación del algoritmo siempre indicará que la firma es verdadera para el mensaje original.

- **Cegera:** Sea $(m, s)$ la salida del protocolo donde $m$ es el mensaje y $s$ la firma resultante, $V$ el conjunto de todos los valores y parámetros a los que tiene acceso el firmante o cualquier otra parte observando el proceso de comunicación entre el firmante y el solicitante de la firma. En cualquier momento el firmante no deberá poder establecer ninguna conexión entre $V$ y $(m, s)$, es decir que es improbable relacionar cualquier par $(m, s)$ a las circunstancias en las que la firma fue construida.

- **Inforjabilidad:** Un esquema de firma digital ciega es inforjable si no importando cuando fue divulgada la firma, el firmante desconoce la identidad del poseedor de la firma. De la misma manera, para un atacante, la única manera de obtener una firma válida es seguir el protocolo con un firmante que posea la llave privada válida.

**Esquema de firma digital ciega RSA-Schnorr**

El siguiente es un protocolo de firma digital ciega que combina los respectivos esquemas de firma ciega RSA y Schnorr [13]. El protocolo se ejecuta de la manera siguiente:

1. El firmante B escoge un número primo $P$ tal que $P = 2pq+1$ donde $p, q$ son dos números primos grandes y diferentes. B publica $P$ y mantiene $p, q$ en secreto. Sea $g$ un generador del grupo de los enteros con la multiplicación módulo $P$, y $e$ un exponente público que verifique que $mcd(e, \phi(P-1)) = 1$ (dónde $\phi$ es la función indicatriz de Euler). Las llave secretas de B son $(p, q, x)$ con $x \in \{0, ..., P-1\}$ y $d$: $d \equiv \frac{1}{e}\ mod\ \phi(P-1)$. Sus llaves públicas son $(P, g, e, y)$ con $y$ tal que $y \equiv g^{-x}\ mod\ P$.

2. El solicitante de la firma A quiere que le sea firmado un mensaje $m$ sin que este sea revelado. Entonces B selecciona un número aleatorio $k \in \{0, ..., P-1\}$ y calcula $r \equiv g^k\ mod\ P$ el cual es enviado a A. Consecutivamente, A escoge 2 números aleatorios $\alpha, \beta \in \{0, ..., P-1\}$. Luego oculta el mensaje a firmar de modo tal que $r' \equiv r^{\alpha^e} y^{-\beta}\ mod\ P$ y calcula el valor $z' \equiv H(m, r')\ mod\ P-1$ y envía el valor $z \equiv (z' + \beta) * \alpha^{-e}\ mod\ P-1$ al firmante B.

3. B firma $z$ con la ecuación $s \equiv (k + z)^d\ mod\ P-1$ y envía el resultado a A.

4. A calcula $s' \equiv \alpha s\ mod\ P-1$ que es la firma final.

5. Cualquiera puede verificar la validez de la firna $(z', s')$ al comprobar si $g^{s'^e} y^{z'} \equiv r'\ mod\ P$ o no.

Siendo los pasos correspondientes a las funciones *Stp*, *Blnd*, *Sig*, $\mathcal{B}lnd^{-1}$ y *Verify* en ese orden. De este modo, los algoritmos quedan como se ven en el algoritmo 3.3.1:

**Algoritmo 3.3.1** Esquema de firma digital ciega RSA-Schnorr

**Función** $Stp(P, e, g = None, x = None)$: ▷ $x, g$ opcionales
  **Si** $g$ es $None$ **entonces**:
    $g \in \{0, ..., P-1\}$ ;
  **Si** $x$ es $None$ **entonces**:
    $x \in \{0, ..., P-1\}$;
  $d = modinv(e, \phi(P-1))$; ▷ Suponiendo que $mcd(e, \phi(e, P-1)) = 1$
  $y = g^{-x} \ mod \ P$;
  secretas $= (x, d)$;
  públicas $= (P, g, e, y)$;
  **regresa** (secretas,públicas);

**Función** $Blnd(m, P, e, y, r)$: ▷ Suponiendo que se hizo la solicitud y se obtuvo $r$
  $\alpha, \beta \in \{0, ..., P-1\}$;
  $r' = r^{\alpha^e} y^{-\beta} \ mod \ P$;
  $z' = H(m, r') \ mod \ P - 1$;
  **regresa** $z = (z' + \beta) * \alpha^{-e} \ mod \ P - 1$;

**Función** $Sig(z)$:
  **regresa** $(k + z)^d \bmod P$;

**Función** $\mathcal{B}lnd^{-1}(s)$:
  **regresa** $\alpha s \bmod P - 1$;

**Función** $Verify(z', s')$:
  **regresa** $g^{s'^e} y^{z'} \bmod P$;

## 3.4. Algoritmo asimétrico NTRUEncrypt

NTRUEncrypt es un algoritmo de criptografía asimétrica (i.e. criptografía de llave pública/privada) basado en retículos, este es un caso especial de NTRU, el cual es conocido por no ser actualmente vulnerable a ataques de computadoras cuánticas (siendo finalista en la tercera etapa del proyecto *Post-Quantum Cryptography Standardization*[5] efectuado por el *National Institute of Standards and Technology* (NIST) del Gobierno de los Estados Unidos).

En NTRU es necesario especificar 3 parámetros para su uso ($N$,$p$,$q$), sin embargo,

para NTRUEncrypt son necesarios 4 $(N,p,q,d)$, estos parámetros son públicos y deben cumplir que:

- $N$ es primo y todos los polinomios deben tener grado estrictamente menor a él.
- $p,q$ deben ser coprimos, $q > p$ y $q > (6d + 1)p$.

La información sobre los algoritmos siguientes puede se encontrada en [8] y en [27]. Para la generación de la llave privada. El creador de la llave $B$ debe especificar 2 polinomios $f,g$, además de los parámetros anteriores, estos polinomios deben cumplir que:

- Su grado es estrictamente menor a $N$.
- Sus coeficientes deben estar en $\{-1, 0, 1\}$.
- $f$ es tal que existen inversos $f_p, f_q$ módulo $p$ y módulo $q$ respectivamente.
- En $f$, el coeficiente "1" y el coeficiente "-1" deben aparecer $d+1$ y $d$ veces respectivamente.
- En $g$, los coeficientes "1" y "-1" aparecen un número igual de veces, siendo $d$ este número.

Una vez teniendo todos estos parámetros en orden, se genera la llave pública siguiendo dos pasos:

1. Se encuentran los inversos $f_p, f_q$ usando el Algoritmo Euclidiano Extendido.
2. Entonces $h = [f_q g \ mod(x^N - 1)] \ mod \ q$ es la llave pública. Aquí $x^N - 1$ es el polinomio de grado $N$ con coeficientes "0" y coeficientes "1" en el término de mayor y menor exponente.

Para encriptar un mensaje con la llave púbica $h$, se procesa el mensaje para convertirlo en un polinomio $m$ con coeficientes binarios. También se genera un polinomio

aleatorio $r$ de coeficientes ternarios con la finalidad de agregar ruido al mensaje encriptado. El mensaje encriptado está dado por:

$$e = [prh + m \ mod(x^N - 1)] \ mod \ q \tag{3.2}$$

Para desencriptar el mensaje $e$ se siguen los pasos siguientes:

1. Se calcula $a(x) \equiv [fe \ mod(x^N - 1)] \ mod \ q$.

2. Se calcula la elevación centrada en $q$ del resultado anterior $a'(x)$.

   - Si $a(x) \in \mathbb{Z}_q[x]$, se define la elevación centrada en $q$ de $a(x)$ como el único polinomio $a'(x) \in \mathbb{Z}[x]$ tal que satisface $a'(x) \equiv a(x) mod \ q$, cuyos coeficientes se encuentran en el intervalo $[-\frac{q}{2}, \frac{q}{2}]$.

3. Finalmente se calcula $[f_p a'(x) \ mod(x^N - 1)] \ mod \ p$, obteniendo como resultado es $m$.

Se representa un polinomio $P = a_k x^x + a_{k-1} x^{k-1} + ... + a_1 x + a_0$ como el arreglo $p = [a_0, a_1, ..., a_{k-1}, a_k]$, donde $p[i]$ representa el coeficiente $a_i$. A $\boldsymbol{p}$ le llamaremos, la **representación en arreglo del polinomio $\boldsymbol{P}$**. De esta manera, los algoritmos de este esquema se pueden ver en el algoritmo 3.4.1.

**Algoritmo 3.4.1** Algoritmo de llave pública/privada NTRUEncrypt

**Función** $NTRU_{keygen}(f = None, g, (N, p, q, d))$: ▷ En caso de tener $f$ y $g$ precalculados, de lo contrario se pueden generar de manera aleatoria.
    $f_p \leftarrow modinv(f, p)$;
    $f_q \leftarrow modinv(f, q)$;
    **regresa** $[f_q * g \ mod(x^N - 1)] \ mod \ q$;

**Función** $NTRU_{encrypt}(mensaje, h, (p, q, N))$:
    $m \leftarrow arreglo(bin(mensaje))$;
    $r \leftarrow randomPoly(\{-1, 0, 1\}, N)$; ▷ Polinomio aleatorio con coeficientes ternarios y grado menor que $N$
    **regresa** $[p * r * h + m \ mod(x^N - 1)] \ mod \ q$;

**Función** $NTRU_{decrypt}(e, f, g, (N, p, q))$:
    $a \leftarrow [f * e \ mod(x^N - 1)] \ mod \ q$;
    $a' \leftarrow elevacion(a, q)$;
    **regresa** $[f_p * a' \ mod(x^N - 1)] \ mod \ p$;

## 3.5. Sistemas biométricos

Un sistema biométrico es un sistema de reconocimiento en el que la identidad de un individuo está determinada a partir de alguna de sus características fisiológicas o de comportamiento. Estos sistemas presentan una mayor fiabilidad en la identificación de una persona ya que los rasgos biométricos no se pierden, no se olvidan y no se pueden compartir.

Un sistema de reconocimiento biométrico se divide en 3 módulos básicos:

- Un **modulo de inscripción** formado por un sistema de adquisición encargado de proporcionar la señal biométrica que caracteriza al individuo. De esta señal se extraen las características del rasgo biométrico del individuo. Estas características forman el llamado *patrón biométrico*.
- Una **base de datos** encargada de almacenar los patrones biométricos de todos los usuarios registrados en el modulo de inscripción

- Un **modulo de reconocimiento** que se encarga de establecer la identidad del individuo que accede al sistema, mediante la adquisición de un rasgo biométrico, la extracción de las características y se obtiene nuevamente el patrón biométrico que posteriormente se compara con los patrones almacenados en la base de datos verificando la identidad del individuo en función del grado de similitud obtenido.

Dado que en este trabajo se utilizará como rasgo biométrico la huella dactilar, se desarrollará la teoría sobre los algoritmos dedicados al reconocimiento de este rasgo. Estos algoritmos siguen los siguientes pasos:

1. Obtención de los patrones directamente de la huella mediante un sensor físico.

2. La imagen obtenida se somete a filtros y operaciones que permiten obtener un vector de patrones que depende de la entrada.

3. Los patrones se analizan y se comparan con los ya conocidos. Ya sea bien usando los patrones de puntos (minucias extraídas de la estructura de las crestas), la textura de la imagen de la huella, o bien la estructura de crestas y valles al completo.


Figura 3.1: Huella dactilar indicando crestas y valles

En la propuesta de este trabajo se utilizarán los algoritmos de identificación de huella digital propios de algún dispositivo de reconocimiento de este rasgo biométrico no solo para la autenticación del usuario, sino que se dará uso también al vector de rasgos, obtenido de dicho aparato, como llave privada.

## 3.6. *Blockchain*

Es una estructura de datos que consiste de una lista retro ligada y ordenada de bloques de transacciones, vigilada constantemente por un conjunto de nodos especializados que la validan dependiendo del protocolo de consenso de la *Blockchain*.

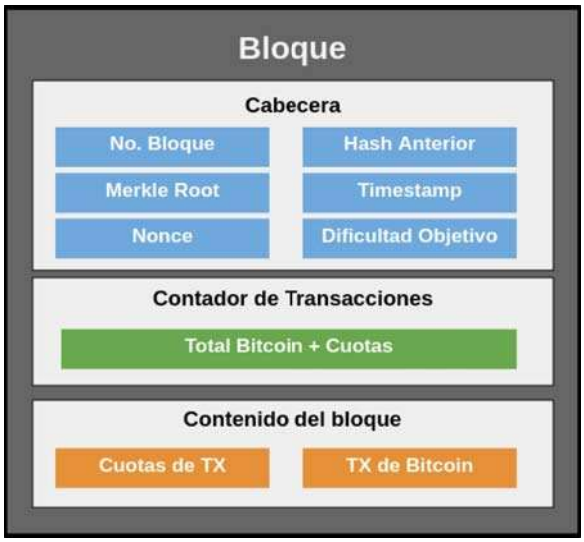



Figura 3.2: Bloque de Bitcoin

De una *Blockchain* podemos resaltar 5 cosas sobre las que desarrollaremos información: una serie de bloques que componen la estructura y guardan la información, el árbol Merkle para proteger dicha información y organizarla usando una función *hash*, un algoritmo de consenso para que exista un acuerdo de veracidad de la información, un algoritmo de encriptación mediante firmas digitales con el que se dará mayor protección a la información y se podrán descartar falsificaciones en caso de necesitarse, y un sistema de *wallets* deterministas para que cada usuario pueda interactuar con la red protegiendo su anonimato.

- Un **bloque** es un objeto informático conformado por datos como el tamaño del bloque, las transacciones guardadas en él, un contador de transacciones y una cabecera: que contiene información tal como el número de bloque, el hash del bloque anterior, la raíz del árbol Merkle, generado por el contenido del mismo mediante la función hash, una marca de tiempo, y dependiendo del protocolo de consenso, un *nonce* y una dificultad objetivo, que serán explicados más adelante.[2] En la figura 3.2 tenemos de ejemplo un bloque de Bitcoin. Bitcoin usa un protocolo de consenso *Proof of Work*(PoW) por lo que en su cabecera incluye los campos de *nonce* y *dificultad objetivo*, además de que se toman en cuenta

aparte las cuotas a pagar por transacción, que serán la recompensa del minero que logre minar el bloque. Se dice que se **mina** un bloque cuando es agregado un bloque a la cadena siguiendo el protocolo de consenso, dentro del cual se llenan los apartados del bloque siguiendo unas reglas predefinidas, de modo que todos los mineros estén de acuerdo que el bloque agregado es válido y contiene información confiable sobre las transacciones y la actividad en el sistema.

- La raíz del **árbol Merkle** sirve como un resumen de los valores hoja contenidos en dicho árbol, éste proceso será explicado a detalle en la sección 3.6.1. La finalidad de realizar éste resumen es que, a largo plazo, con miles y miles de transacciones realizadas por los usuarios, el espacio para almacenar una *Blockchain* completa podría convertirse en un problema, sin embargo, "una vez que una transacción está encerrada bajo suficientes bloques, las transacciones gastadas con anterioridad pueden ser descartadas para ahorrar espacio en disco. Bloques viejos pueden ser compactados desprendiendo las ramas del árbol. Hashes anteriores no necesitan ser almacenados" (Nakamoto, Satoshi, 2008) [23]

- Una estructura *Blockchain* provee una manera en que los participantes concurran en el estado global de la cadena de una manera *peer-to-peer*. A esto se le llama **algoritmo de consenso** y se profundizará más sobre el tema en la sección 3.6.2.

- Se define una criptomoneda como una cadena de **firmas digitales**. Debido a esto, es importante que la *Blockchain* haga uso de un sistema de firma digital de llave pública y privada, que serán utilizadas para encriptar la información referente a validación, a fin de evitar falsificaciones.

- Una **wallet** es una estructura de datos usada para almacenar y administrar las llaves de un usuario (ver sección 3.6.3). Existen dos tipos, las aleatorias o *no-deterministas* que generan colecciones de llaves privadas aleatoriamente, por otro lado están las *deterministas* que generan llaves privadas que son derivadas de una *semilla* común, y el uso de una función hash criptográfica de un solo

sentido.

Las primeras presentan demasiados problemas para los usuarios y un peligro para el sistema, ya que al ser necesario que ninguna llave sea repetida por los usuarios, es necesario tener un respaldo y registro de todas las llaves generadas, lo que las convierten en una carga muy grande para la red. Por esta razón se utilizarán y desarrollará información sobre las segundas.

Cada bloque es identificado por un hash, generado a partir de su contenido, con esto cada bloque está ligado a su padre al referenciar el hash del bloque anterior creando así una cadena. Cada bloque padre puede tener temporalmente múltiples hijos (los cuales hacen referencia al mismo padre), dando a una bifurcación que sucede cuando varios nodos generan casi simultáneamente un bloque. Esta bifurcación se resuelve mediante la prueba de consenso de la *Blockchain*, dejando únicamente a un bloque padre con un bloque hijo.

De esta manera, podemos definir que una *Blockchain* en el nodo $i$ de la red es $\langle B_i, P_i \rangle$, un grafo acíclico dirigido con bloques $B_i$ y apuntadores $P_i$.

### 3.6.1. Árbol Merkle

Un árbol de Merkle, también conocido como árbol de Hash se trata de una estructura de datos en árbol usada para resumir y verificar grandes cantidades de datos. En un árbol de Merkle cada hoja contiene un hash criptográfico, y es construido recursivamente de abajo hacia arriba al aplicar la función hash a pares de nodos hijos, fusionándolos en el nodo padre hasta que queda un único hash, llamado **la raíz**, o **raíz de Merkle**.

La construcción de un árbol Merkle es bastante sencilla. Como se dijo anteriormente, es construido comenzando por las hojas, de este modo, a cada dato (en este caso una transacción) le es aplicada la función hash dos veces para construir la hoja en la que se guardará. Por ejemplo, sea $A$, una transacción, y $H_A$ la hoja en la que

será almacenada. $H_A$ está dada por:

$$H_A = hash(hash(A))$$

Sucesivamente, pares consecutivos de nodos hoja son fusionados luego en un nodo padre, al concatenarlos y aplicar la funcion hash a dicho resultado. Por ejemplo, para construir el nodo padre nodo $H_{AB}$, los dos códigos hash de los nodos hijos se concatenan para crear una cadena de caracteres. A esa cadena se le aplica un hash doble para producir el hash del nodo principal:

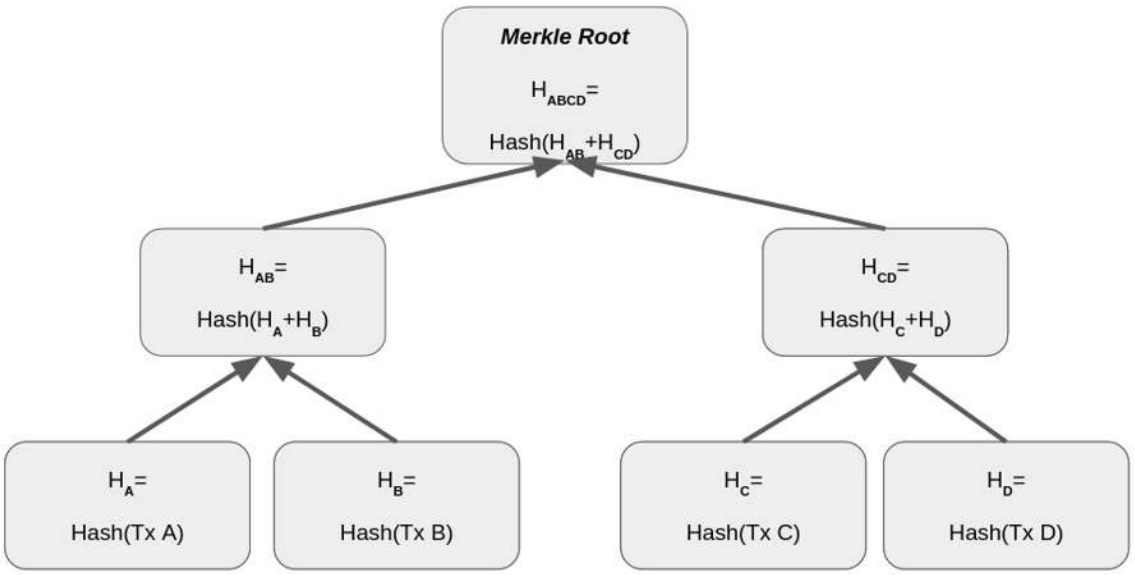
$$H_{AB} = hash(hash(H_A + H_B))$$

Este proceso es continuado hasta obtener un único nodo en la cima como se puede observar en la figura 3.3 a), dicho nodo es conocido como la raíz del árbol Merkle. Debido a que el algoritmo trata con datos emparejándolos, cuando solo se tiene una cantidad impar de datos, el último dato es duplicado, como se muestra en la figura 3.3 b), para obtener un árbol balanceado con un número par de datos.

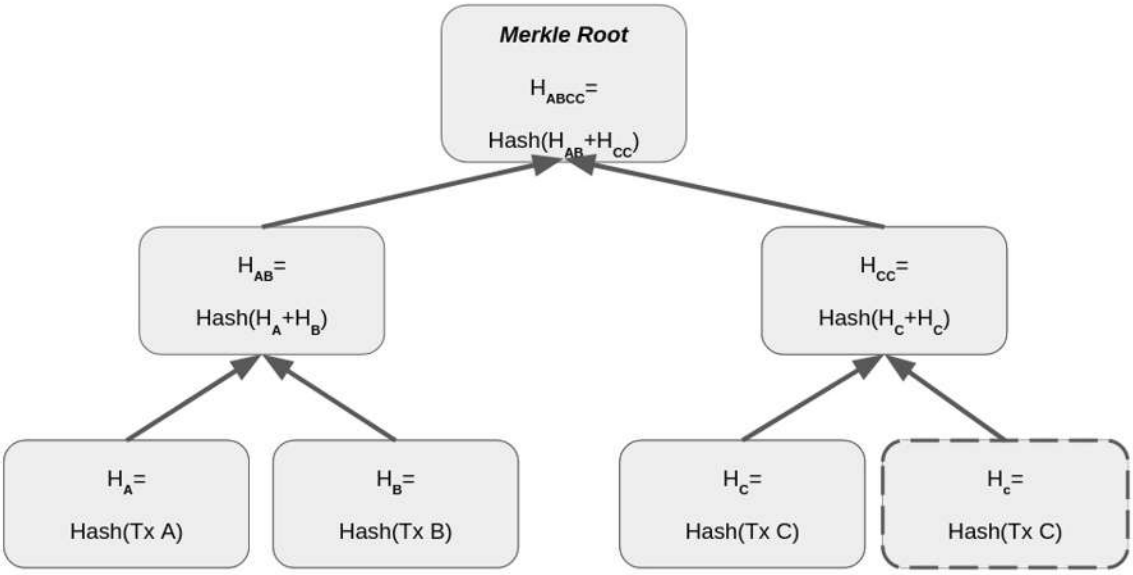
Para un árbol con N hojas un algoritmo puede computar caminos de autenticación de datos en tiempo $2 * log_2(N)$ y espacio menor que $3 * log_2(N)$, donde las unidades de cálculo son evaluaciones de funciones hash o cálculos de valores de hoja, y las unidades de espacio son el número de valores de nodo almacenados. De lo cual podemos observar que esta es una estructura de datos muy eficiente.[30]

Entonces, para encontrar una transacción específica en un bloque de la *Blockchain*, un nodo solo necesita producir $log_2(N)$ códigos hash, constituyendo un *camino de autenticación* o *camino de Merkle* conectando la transacción específica con la raíz, esto es especialmente importante dado que mientras las transacciones pueden crecer de una manera muy acelerada, el logaritmo de esta cantidad crece muy lentamente.

El pseudocódigo 3.6.1 muestra con detalle la manera en que funciona y se programa un árbol de Merkle, a partir de una lista de códigos hash “hashList”:




**a)**




**b)**

Figura 3.3: a) Construcción de un árbol Merkle; b) Manejo de datos impares

**Algoritmo 3.6.1** Árbol de Merkle

```
1: Función CREAARBOL(hashList):
2:     Si hashList.vacia entonces:                    ▷ Parar si la lista está vacía
3:         regresa null;
4:     si no, Si hashList.tamaño = 1 entonces:
5:         regresa hashList[0];
6:                                                    ▷ Iterar sobre la lista si no es vacía
7:     Mientras hashList.tamaño>1 hacer               ▷ Si es impar se duplica el último
   elemento en la cola
8:         Si (hashList.tamaño mod 2) ≠ 0 entonces:
9:             hashList.push(hashList[-1]);
10:
11:        nuevaLista;                                ▷ Se crea una nueva lista
12:        Para cada 2 hash en hashList hacer
13:            H_padre = H_1 + H_2
14:            nuevaRaíz = hash(hash(H_padre))
15:            hashList.push(nuevaRaíz)
16:
17:        hashList = nuevaLista
18:    regresa hashList[0];          ▷ El algoritmo devuelve la raíz del árbol creado
```

### 3.6.2. Protocolos de consenso

El **consenso** es un problema bien conocido de la informática distribuida. Consiste en lograr un acuerdo entre un número distribuido de procesos [4]. A diferencia de otras estructuras ligadas, *Blockchain* provee un método de asegurar que existe una sola versión de la cadena. A esta solución se le conoce como algoritmo (o protocolo) de consenso. Estos algoritmos combinan criptografía y algún tipo de incentivo para hacer cumplir el objetivo de que la cadena sea correcta e inmutable. Además, se asegura que el siguiente bloque de la cadena es la única versión de la verdad, y previene que otros adversarios descarrilen el sistema y forjar una cadena impostora.

Entre otros, un esquema de consenso destacado es el llamado *Byzantine Fault Tolerance (BFT)* basado en El Problema de los Generales Bizantinos [22]. Los protocolos de este tipo pueden tolerar nodos subvertidos arbitrariamente que intentan obstaculizar el logro de un acuerdo consistente. En las primeras soluciones a dicho

problema el número requerido de nodos subvertidos $n$ con el que existía solución era de $n < \frac{M}{3}$ donde $M$ es el número total de nodos. No obstante, en las nuevas soluciones, el número requerido para que el problema no tenga solución está dado por $n > \frac{M}{2}$.

Existe una gran cantidad de familias de algoritmos de consenso del tipo BFT, es decir basadas en dicho problema, no obstante solo se tratarán 2.

***Delegated Proof-of-Stake y Proof-of-Authority***

En determinados sistemas basados en *Blockchain*, siempre se puede sacrificar un poco la descentralización, con la finalidad de obtener un consenso más eficaz y confiable, además de un mejor rendimiento en el funcionamiento de la red.

Tal es el ejemplo de aquellos sistemas que utilizan como algoritmo de consenso protocolos de las familias llamadas *Delegated Proof of Stake*, y *Proof of Authority*. El primero es una forma especializada del protocolo de consenso *Proof-of-Stake*, en el que el grueso de los participantes de la red no participan en la votación de la validación de los bloques, en su lugar, la intención es que esta responsabilidad sea delegada a un grupo severamente limitado de participantes. De este hecho surge su nombre, y los participantes escogidos son llamados *delegados*.

Por otro lado, *Proof-of-Authority* (PoA) es una familia de algoritmos de consenso que dependen de un conjunto de nodos de confianza denominados *autoridades*. Cada autoridad está identificada por un identificador (id) único y la mayoría de ellas se asume honesta, es decir, al menos $\frac{n}{2} + 1$. Las autoridades llegan a un consenso para ordenar las transacciones emitidas por los clientes. El consenso en los algoritmos de PoA se basa en el esquema de rotación de minería, un enfoque ampliamente utilizado para distribuir equitativamente la responsabilidad de la creación de bloques entre las autoridades. El tiempo se divide en pasos, cada uno de los cuales tiene una autoridad elegida como líder minero [16].

En esta sección tomaremos como referencia el algoritmo Clique, usado en *Blockchains* privadas de Ethereum que pertenece a la familia PoA.
Un bloque en una *Blockchain* que utiliza un protocolo PoA reemplaza los apartados de *nonce* y *dificultad objetivo* en el header de la figura 3.2 por otros que llevan por nombre *sealer* y *weight* (o bien, *sealer* y *step*, u otros dependiendo de la implementación). Estos guardan el id de la autoridad que firmó el bloque y el "peso" del bloque (que se definirá más adelante) en el que fue agregado el bloque a la cadena. Se utilizará el término *padre* para representar al hash del bloque anterior, de esta manera definimos un bloque como:

$$b = (number, Timestamp, padre, MerkleRoot, sealer, weight) \tag{3.3}$$

En el algoritmo 3.6.2 se ilustra el algoritmo de Clique, y servirá de ejemplo para describir como funciona un algoritmo de esta familia. En este algoritmo cada nodo autoridad de la red ($p_i : i \in Nodos$) comparte el bloque génesis en el que está escrito el *periodo*, el lapso de tiempo entre la creación de bloques consecutivos. Cada nodo $i$ mantiene su propia copia de la *Blockchain* como un grafo acíclico dirigido $\langle B_i, P_i \rangle$ y mantiene un estado constante en el que corre un bucle infinito (función "proponer()") que ejecuta el protocolo y permite al nodo forjar un bloque cuando se cumplen las condiciones escritas dentro del bucle [12].

La primera de las condiciones para poder firmar (línea 18) es esperar a que los demas *sealers* firmen bloques hasta que ninguno de los últimos $\lambda = limite$ bloques contenga la firma del *sealer* $i$. En esta implementación el $limite$ es $\lfloor \frac{|sealers|}{2} \rfloor + 1$ que es la mínima mayoría. La siguiente condición (línea 20) es esperar a que pase el tiempo necesario entre que se forjó el último bloque para poder forjar el siguiente. Si ambas condiciones se cumplen concurren, se procede a comprobar si es turno del *sealer* $i$ para firmar, de ser así el bloque se firma con $peso = 2$, de lo contrario se firma el bloque con $peso = 1$ despues de un retraso aleatorio entre 0 y $500 \times \lfloor \frac{|sealers|}{2} \rfloor + 1$ milisegundos. El último paso en el bucle es "distribuir()" la copia de la cadena del

nodo $i$ a los demás nodos autoridad.

Al recibir un nodo una copia de una cadena, la funcion "actualiza()" es invocada, usando la función "pesoTotal()" para comparar el peso de la copia local de la cadena en el nodo con el peso de la cadena recibida. Este proceso actualiza la copia local de la cadena en caso de que la recién recibida sea más pesada, es decir sus bloques fueron correctamente firmados por los nodos en sus turnos correctos. De lo contrario mantiene la copia que poseía.

El consenso respecto a un bloque es alcanzado una vez que un bloque $b$ está "decidido()", esto es si el orden del conjunto de *sealers* que han firmado y agregado bloques a la cadena después del bloque $b$ es mayor a la mínima mayoría, es decir $\lfloor \frac{|sealers|}{2} \rfloor + 1$.

**Algoritmo 3.6.2** Pseudocódigo PoA en el nodo $p_i$

1: $sealers \subseteq Nodos$; ▷ El conjunto de autoridades
2: $c_i = \langle B_i, P_i \rangle$; ▷ La copia de la *Blockchain* en el nodo i
3: $b$; ▷ Un bloque vacío como en la ecuación 3.3
4: $periodo$; ▷ La duración mínima en segundos entre timestamps de dos bloques consecutivos
5: $mayoria \leftarrow \lfloor \frac{|sealers|}{2} \rfloor + 1$;
6: $limite \leftarrow mayoria$ ▷ El número máximo de bloques consecutivos en el que un *sealer* puede firmar a lo más 1 bloque
7:
8: **Función** FIRMADORECIENTE($c_i, n$):
9: $\lambda \leftarrow limite$;
10: $ret$= false;
11: **Para** m=n-$\lambda$,...,n **hacer**: ▷ Iterar sobre los últimos $\lambda$ bloques
12: **Si** $b_m.number$ mod$|sealers| = i$ **entonces**:
13: $ret$=True;
14: **regresa** $ret$;
15:
16: **Función** PROPONER( ):
17: **Mientras** True **hacer**:
18: $n \leftarrow$ últimoBloque($c_i$).$number$; ▷ Índice del último bloque
19: **espera hasta que** firmadoReciente($c_i, n$)=False ▷ Espera hasta que pueda firmar un bloque
20: $T \leftarrow b_n.Timestamp$;
21: **espera hasta** clock $\geq T + periodo$ ▷ La espera sea mayor que el periodo
22: **Si** $(n+1)$mod $|sealers| = i$ **entonces**: ▷ Turno del nodo $p_i$
23: $b.peso$=2;
24: **si no,** ▷ Sellado fuera de turno
25: sleep(rand([0,500×$mayoria$])); ▷ Retraso aleatorio en ms
26: $b.peso$=1;
27: $b.number = n + 1$;
28: $b.padre \leftarrow$ últimoBloque($c_i$);
29: $b.sealer \leftarrow$ firma(); ▷ Se firma el bloque $b$
30: $c_i \leftarrow \langle B_i \cup \{b\}, P_i \cup \{b.padre\} \rangle$; ▷ Se encadena el bloque
31: distribuir(); ▷ Se distribuye el bloque
32:
33: **Función** PESOTOTAL($\langle B_j, P_j \rangle$):
34: **regresa** $\Sigma b.pesos : b \in B_j$;
35:
36: **Función** ACTUALIZA($\langle B_j, P_j \rangle$):
37: **Si** pesoTotal($\langle B_j, P_j \rangle$)> pesoTotal($\langle B_i, P_i \rangle$) **entonces**: $\langle B_i, P_i \rangle \leftarrow \langle B_j, P_j \rangle$ ▷ Se actualiza la cadena por la más pesada
38:
39: **Función** DECIDIDO($b$):
40: $V \leftarrow \{b_k.sealer : b_k \in B; k \geq i\}$
41: **regresa** $|V| > mayoria$;

### 3.6.3. Wallet determinista

Se define una **wallet** como una estructura de datos utilizada para almacenar y administrar las claves de un usuario. [2] Existen principalmente dos tipos de wallets distinguidas entre sí por como se relacionan las claves que contienen o por el contrario si no lo hacen.

El tipo conocido como wallet no determinista, donde cada llave se genera de forma independiente a partir de un número aleatorio. Las llaves no están relacionadas entre sí. El otro tipo de wallet es llamado determinista, donde todas las claves se derivan de una única llave maestra, conocida como semilla. Todas las claves de este tipo de billetera están relacionadas entre sí y se pueden generar de nuevo si se tiene la semilla original. Hay un número de diferentes métodos de derivación de claves utilizados en carteras deterministas. El más comúnmente utilizado usa una estructura en forma de árbol y se conoce como **wallet jerárquica determinista** o HD (*hierarchical deterministic*).

Las HD wallets contienen llaves derivadas en una estructura de árbol, de modo que una llave padre puede derivar una secuencia de llaves hijas y así sucesivamente. Una de las ventajas más destacables de las HD wallets es que los usuarios pueden crear una secuencia de llaves públicas sin tener acceso a las correspondientes llaves privadas, lo cual protege al usuario de poder revelar la llave privada por accidente. Además, las llaves públicas no necesitan estar precargadas ni ser derivadas de antemano, más aún ningún servidor tiene registro de las llaves privadas de modo que pueda hacerse un mal uso de éstas.

El procedimiento usado en bitcoin para hacer una wallet determinista es el siguiente, en primer lugar se crea una wallet determinista con un código nemotécnico, a partir de este código se genera la semilla para crear la HD wallet. Los pasos a seguir son los siguientes [26], y son ilustrados en el algoritmo 3.6.3:

1. Se crea una secuencia aleatoria (entropía) de 128 a 256 bits.
2. Se aplica la función hash a la secuencia aleatoria y se toman los primeros $\frac{tam(Entropia)}{32}$ bits y se crea una suma de verificación usando esos bits tomados, con la finalidad de detectar cambios en la secuencia de datos para proteger su integridad y que no haya discrepancias.
3. Se agrega la suma de verificación al final de la secuencia aleatoria.
4. Se divide la entropía en secciones de 11 bits.
5. Para cada una de estas secciones se toma la palabra correspondiente a su valor de un diccionario predefinido de tamaño 2048 y se agrega al código nemotécnico.
6. El código nemotécnico es la cadena de palabras.
7. Se toma el código nemotécnico y una *salt*, la cual es una cadena (o una contraseña) cuyo propósito es que sea más difícil reproducir una semilla usando solo el código nemotécnico.
8. Usando la función PKDF2 y el algoritmo hash criptográfico HMAC-SHA512 se crea la semilla para crear la HD wallet.

**Algoritmo 3.6.3** Generacion de semilla a partir de código nemotécnico

```
1:  entrp ← generaEntropia();
2:
3:  Función NEMOTÉCNICA(entrp):
4:      aux ← SHA256(entp);
5:      checksum ← aux[0 : (len(entrp)/32)];
6:      nuevo ← entrp + checksum;
7:      divdd ← Split(nuevo, 11);
8:      código;
9:      Para cada parte en divdd hacer:
10:         palabra ← dict(parte);
11:         código += " " + palabra;
12:     regresa código;
13:
14: nemotecnico ← Nemotecnica(entrp);
15: salt ← obtenSalt();
16: semilla ← PKDF2(nemotecnico, salt, HMAC-SHA512);
```

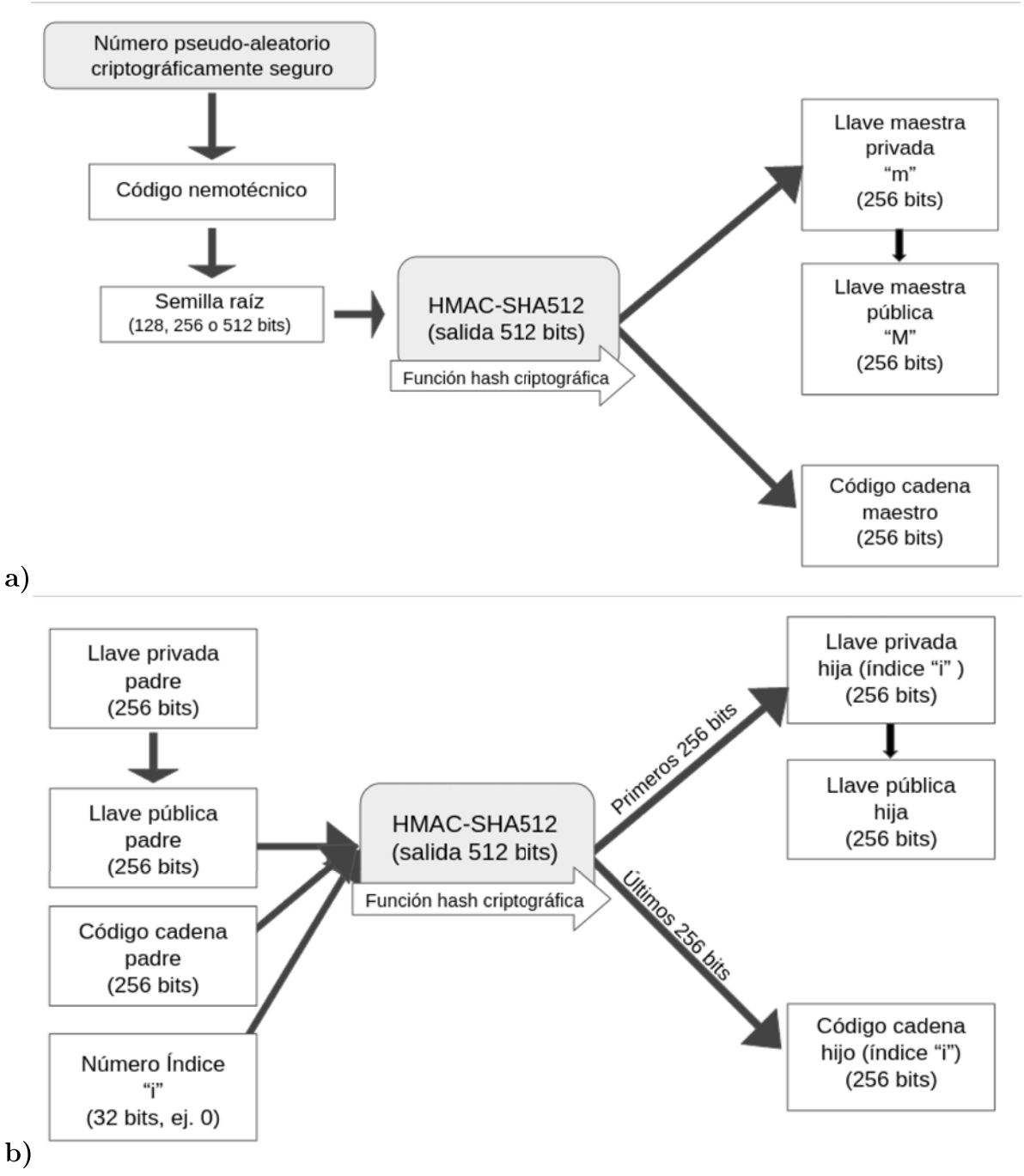



Figura 3.4: a)Generación de las llaves principales. b)Derivación de las llaves hijas

Una vez generada la semilla raíz, se procede a crear las llaves principales de la wallet, bajo los siguientes pasos y se ilustra en la figura 3.4 a):

1. Usar la función hash criptográfica HMAC-SHA512 tomando como entrada la semilla raíz.

2. Dado que la salida de la función son 512 bits, se toman los primeros 256 bits y estos serán la llave maestra privada.

3. Usando el esquema de encriptación llave pública-privada escogido, se deriva la llave maestra pública a partir de la llave maestra privada del paso anterior.

4. Los 256 bits restantes no usados en el paso 2 serán llamados código cadena maestro.

A partir de la llave principal se procede a la creación de llaves derivadas para generar la estructura jerárquica, el procedimiento que se sigue para ello es el siguiente y es ilustrado en el algoritmo 3.6.4 y la figura 3.4 b). :

1. Se concatenan la llave pública padre, el código cadena padre y un número que indique el número de llave hija a generar en el nivel.

2. Se utiliza la función hash criptográfica HMAC-SHA512 con la combinación anterior de entrada, para producir un hash de 512 bits.

3. Al igual que en el proceso de generar las llaves principales, con el hash generado se toman los primeros 256 bits para crear una llave privada hija que servirá para generar una llave pública hija.

4. Así mismo, los 256 bits restantes se usan para crear un código cadena hijo que servirá para crear más llaves hijas utilizando la nueva llave pública hija del paso anterior.

En este proceso es fundamental el código cadena para introducir datos aleatorios en el mismo, de modo que no sea suficiente tener posesión de la llave pública padre y el índice para derivar las llaves hijas. Así mismo, tener el conocimiento de una llave

hija no es suficiente para encontrar las llaves hermanas, a menos de que se tenga el código cadena.

**Algoritmo 3.6.4** Creación de la estructura de arbol de llaves en una HD wallet

```
1: Función llavesHijas(llavePadre, códigoCadenaPadre, índice):
2:     input ← llavePadre+códigoCadenaPadre+índice;
3:     hash ← HMAC-SHA512(input);
4:     nuevaPrivada ← hash[0:255];
5:     nuevoCódigoCadena ← hash[256:512];
6:     regresa (nuevaPrivada, nuevoCódigoCadena);
```

### 3.6.4. Direcciones anónimas

En Bitcoin y otras criptomonedas, una dirección (*address* o hash de llave pública) es una cadena de caracteres que puede ser compartida con cualquiera que quiera enviar alguna cantidad de *tokens* al propietario de dicha dirección. Son generadas a partir de alguna llave pública del usuario y generalmente comienzan con algún prefijo.

Si $P$ es la llave pública del usuario y $pre$ es un prefijo predefinido, el proceso que se sigue para generar una dirección a partir de $P$ es el siguiente y es mostrado en el algoritmo 3.6.5:

1. Se define la variable $hash = RIPEMD160(SHA256(P))$.
2. Así mismo, se define una $check = SHA256(SHA256(pre + hash))$ (donde "+" indica concatenación).
3. Se define *checksum* como los primeros 4 dígitos de *check* definido en el paso anterior.
4. El resultado, $address = BASE58(pre||hash||checksum)$ es la dirección deseada.

**Algoritmo 3.6.5** Direcciones anónimas

1: **Función** ANONDIR($P$,$pre$):
2: $hash = RIPEMPD160(SHA256(P))$;
3: $check = SHA256(SHA256(pre + hash))$;
4: $checksum = check[0:3]$;
5: **regresa** $BASE58(pre + hash + checksum)$;

Aquí BASE58 es un formato de codificación que consiste de las 52 letras del alfabeto inglés (26 mayúsculas, 26 minúsculas), junto con 10 numerales exceptuando caracteres que son idénticos en apariencia y puedan generar confusiones al leerse por el usuario, esto es: el número cero (0), la letra o mayúscula (O), la letra L minúscula (l), y la letra i mayúscula (I). Dando como resultado un alfabeto de 58 caracteres.

En este algoritmo se usan en conjunto el doble hash RIPEMD-160, y SHA-256 para que el resultado final fuera de tamaño 160 bits haciéndolo más corto y manejable, y que sin embargo que sea seguro. Debido a que se han encontrado vulnerabilidades para RIPEMD-160, en esta propuesta se sustituirá el uso de RIPEMD-160 por SHA-1, ya que producen salidas del mismo tamaño y el último reporta ser más veloz que el primero.

## 3.7. Archivos JSON

Un archivo JSON (acrónimo de *JavaScript Object Notation*) es un formato de texto para la serialización de datos estructurados de estándar abierto diseñado para el intercambio de estos. Definido en la tercera edición del ECMAScript Programming Language Standard [1][3]. Un archivo JSON puede representar cuatro tipos de datos primitivos (cadenas de caracteres, números, booleanos y la palabra reservada "null") y dos tipos de datos estructurados (objetos y arreglos). Sus objetivos son ser portátil, textual y mínimo respecto al tamaño.

- Una **cadena de caracteres** (o simplemente **cadena**) es una secuencia de cero o más caracteres Unicode. Gramáticamente una cadena de caracteres será representada entre comillas.

- Un **booleano** es un dato lógico que puede representar uno de los dos valores de lógica binaria representados por las palabras reservadas "true" o "false".

- Un **arreglo** es una secuencia ordenada de cero o más valores de los listados anteriormente. Gramáticamente es representado entre corchetes, y cada valor es separado por una coma, es decir: [valor$_1$, valor$_2$, ... , valor$_n$].

- Un **objeto** es una colección desordenada de cero o más pares (*nombre*,*valor*), donde el *nombre* es una cadena y el *valor* puede ser una cadena, un número, un booleano, un objeto, un arreglo o bien un valor "null"). Gramáticamente cada par (*nombre*, *valor*) es representado sin paréntesis mientras que el *nombre* y *valor* son separados por dos puntos, es decir: *"nombre": valor*. Por otro lado, un objeto es escrito entre llaves y cada par es separado por una coma es decir: {*"nombre$_1$": valor$_1$*, *"nombre$_2$": valor$_2$*, ... , *"nombre$_n$": valor$_n$*}.

    - Para este trabajo, se define una tabla hash como un **objeto** en el cual el *nombre* de cada par es una cadena generada por una función hash que recibe de entrada características relacionadas al *valor* correspondiente.

Gramáticamente, un texto JSON es un valor serializado de alguno de los mencionados en la lista anterior. En el presente trabajo se utilizarán archivos de este formato para el intercambio de información entre nodos de la *Blockchain*, así como entre los usuarios del sistema con los nodos. Dada la naturaleza de la información que se intercambia, los textos JSON utilizados en este trabajo serán de la forma de un objeto.

# Capítulo 4

# Propuesta

En este trabajo se presenta un sistema de votaciones electrónicas que está pensado para llevarse a cabo en primer lugar dentro de algún tipo de institución (universidades, asociaciones, etc.) . Sin embargo, se espera que también pueda ser escalado para ser usado en elecciones federales. A pesar de poder ser implementado para votar de manera remota, en lo subsiguiente se considerará que las votaciones se llevan a cabo de manera presencial en algún recinto indicado por la autoridad.
Las principales propiedades que se esperan cumplir en esta propuesta son aquellas expuestas en [15]:

- **Exactitud:** Si todos los participantes de las elecciones (votantes, autoridades, etc.) son honestos y se comportan como se espera, entonces los resultados finales son efectivamente el recuento de votos emitidos.
- **Privacidad:** La identidad de los votantes debe mantenerse desconocida, y ningún voto debe poder ser relacionado con ningún votante.
- **Ausencia de recibo:** Los votantes no deben poder reconstruir ni obtener un recibo que acredite el contenido de su voto. De modo que se prevenga la venta de votos ni la coerción.
- **Robustez:** El sistema de votación pueda tolerar un cierto número de participantes tramposos ($<50\,\%$) y ataques que atenten contra su funcionamiento.

- **Verificabilidad:** Los procesos de votación correctos deben ser verificables para evitar resultados de votación incorrectos. Existen 2 tipos de verificabilidad a cumplir:
  - *Verificabilidad universal (pública):* Cualquier participante u observador pasivo puede convencerse de la validez de los votos individuales y del conteo final de la elección.
  - *Verificabilidad individual:* Todo votante elegible puede verificar que su voto fue contado.
- **Democracia:** Solo los votantes registrados pueden votar y todos los votantes registrados pueden emitir libremente su voto una sola vez, y no más.
- **Equidad:** Ningún participante puede obtener ningún conocimiento, a excepción de su voto, sobre el conteo (parcial) antes de la etapa de conteo, pues el conocimiento del conteo parcial podría afectar las intenciones de los votantes que aún no han votado.

## 4.1. Visión General

Este sistema consiste en una aplicación multiplataforma para dispositivos electrónicos, y una *Blockchain*, en algún servidor o servidores web perteneciente a la institución organizadora. La aplicación deberá instalarse en algún dispositivo que cuente con un periférico de identificación biométrica mediante digitalización de huellas dactilares. El proceso de esta propuesta está compuesto por 6 etapas, las cuales son:

1. Etapa de Registro de los Votantes
2. Etapa de Prevotación
3. Etapa Autenticación del Usuario
4. Etapa de Votación

5. Etapa de Validación del Voto y Minado de Bloques

6. Etapa de Conteo.

En primer lugar, en la Etapa de Registro, la institución se asegura que todos los votantes están registrados en el padrón y tienen en su base de datos los registros de alguna (o algunas) de sus huellas dactilares, una contraseña que es generada por el usuario al momento de este registro, y la generación de una identificación de pertenencia al padrón, que contenga un código QR, o bien un chip NFC, que contengan en su interior un código nemotécnico que sirva para crear una semilla a partir de la cual pueda ser producida una HD wallet para el usuario, y se crea su primer dirección pública dentro de la red.

En la Etapa de Pre-votación (el día previo al día de votación, o bien, el mismo día antes de la apertura de las casillas) la institución realiza la implementación de la *Blockchain* creando el bloque génesis, en el cual se escribe una transacción a nombre de cada una de las direcciones públicas que tiene registrados en su base de datos, correspondientes a cada uno de los votantes registrados para que estos puedan efectuar su voto.

En la Etapa de Votación, el usuario accede a la aplicación en la que su pantalla de inicio, como la mostrada en la figura 4.1 muestra un recuadro en la parte central superior, en el que deberá escribir su contraseña; del lado inferior izquierdo se encuentra un botón en el que al presionarlo el usuario puede escanear su (o sus) huellas digitales; también en la parte inferior pero del lado derecho, se encuentra un botón igual, pero que al ser presiona-

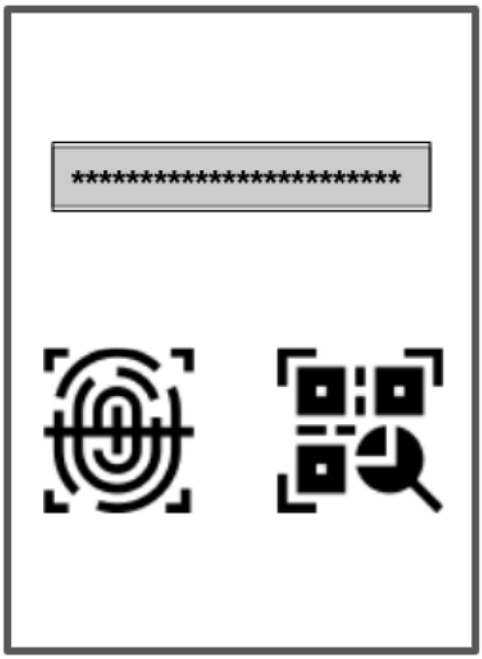



Figura 4.1: Autenticación del usuario

do permite al usuario escanear el código (o bien, el chip NFC) de su identificación. De esta manera se logra un control de autenticación de 3 factores, en el que al usuario se le concede acceso al demostrar que es quien mediante algo que sabe (su contraseña), algo que posee (su identificación), y algo que es (su huella dactilar).

Una vez que la aplicación comprueba la veracidad de la identidad del usuario, la wallet comprobará si el usuario tiene exactamente 1 token en su posición, si lo tiene se mostrará la pantalla de votaciones, en la que aparecerán los candidatos posibles a ser elegidos (tal como en la pantalla de una máquina de votación *DRE*), ahí el usuario podrá hacer uso efectivo de su derecho al sufragio y emitir su opinión. En caso de que no tenga exactamente 1 token, la aplicación indicará un error. En caso de que ocurra algún error en los pasos anteriores, se notificará el error en pantalla para que el usuario vuelva a intentar en caso de error de identificación, o bien notificando que ya ejerció su voto.

Habiendo emitido el voto, la HD wallet genera un árbol de direcciones públicas de tamaño 3, y escribe la transacción en la *Blockchain* que le indicará a la wallet que el usuario gastó su token, no obstante manteniendo la anonimidad del usuario. Por otro lado el voto es enviado a través de Internet y es recibido por el servidor web, que mantiene corriendo el protocolo de consenso de la *Blockchain* que se encarga de validarlo y agregarlo a un bloque que será anexado a la cadena, donde será almacenado. Al final de la jornada electoral, los votos almacenados en la cadena de bloques son contabilizados para realizar el conteo final que será liberado al público.

Este sistema podría clasificarse como *voto por Internet*, no obstante puede ser adaptado fácilmente a un sistema *DRE* de red pública mediante la instalación autorizada de la aplicación en una tableta electrónica o teléfono inteligente que cuente con sensor de huellas dactilares, o bien en una computadora con dicho sensor de manera integrada, o que pueda ser conectado a ésta de manera externa. Éstos serán instalados en lugar de las casillas normales de voto en papel.

## 4.2. El sistema completo

La primera parte a desarrollar para la implementación de este sistema es la programación de la *Blockchain*, que será parte fundamental en todas las etapas pues sus propiedades otorgan las bases para que el sistema de votación sea democrático, verificable, robusto, y mantenga la privacidad de los votantes. El protocolo de consenso que usará sera *Proof of Authority*, y los usuarios interactuarán con ella mediante una HD wallet para mantener su privacidad, junto con el algoritmo de encriptación basado en retículos NTRU, los algoritmos del esquema de firma digital de curvas elípticas FDCE, el esquema de firma digital ciega RSA-Schnorr, además de las funciones hash criptográficas SHA-1, SHA-256 usándola junto con HMAC.
Para comenzar, definimos un bloque de la manera siguiente:

$$Bloque = \{No,\ Hash\ Padre,\ Tm,\ Peso,\ Merkle\ Root,\ Autoridad,\ Votos,\ Dx\} \quad (4.1)$$

Donde:

- $No$ es el número del bloque en la cadena
- $HashPadre$ es el hash del bloque anterior.
- $Tm$ es el *timestamp* en el que fue minado el bloque
- $Peso$ es el peso correspondiente al bloque
- $MerkleRoot$ es la raíz del árbol Merkle generado tomando el hash del bloque anterior, el conjunto $Votos$ y el conjunto de transacciones como hojas.
- *Autoridad* es el nodo autorizado de la *Blockchain* que verificó, autorizó y agregó el bloque.
- $Votos$ es el conjunto de los votos emitidos por los usuarios en texto plano, de modo que puedan ser contabilizados.
- $Dx$ es el conjunto de direcciones anónimas que emitieron su opinión y que fue guardada en este bloque.

El bloque usado en esta propuesta 4.1, difiere del definido en la sección 3.3 únicamente en que fueron agregados los conjuntos de votos y de direcciones anónimas, con la finalidad de que fuera ahí donde se guardara la información importante y relevante para esta propuesta.

Por otro lado, definimos una transacción como un archivo JSON, estructurado de la manera siguiente y cuyo contenido será explicado a profundidad en la sección 4.2.4:

$$\{"Remitente" : "","Voto" : "","Firma" : ""\} \tag{4.2}$$

Una vez definidos el objeto bloque con sus propiedades y el modelo a seguir por las transacciones, definimos la *Blockchain* como una tabla hash de bloques, de este modo se puede acceder a cada bloque por medio de su $MerkleRoot$, con una complejidad promedio de $O(1)$. De manera paralela, se crea el conjunto $Nodos$ que mantendrán la *Blockchain* activa además de ser los que se encargarán de servir como autoridad, también se crea el conjunto de direcciones anónimas que emitieron su voto, una lista de transacciones o *mempool* en la que serán alojadas las transacciones hasta ser escritas en un bloque y una lista enlazada simple cuyo contenido serán los códigos hash identificadores de cada bloque que será una estructura auxiliar que mantenga la estructura de grafo dirigido, propia de la *Blockchain*. Estas estructuras servirán de manera auxiliar al funcionamiento de la cadena.

El pseudocódigo de la clase *Blockchain* puede verse en el algoritmo 4.2.1. La función invocada en las líneas 13 y 25, dentro de la función "creaBloque()" y "esValida()" será la misma función del pseudocódigo 3.6.1 que así mismo debe ser incluida en la clase. Más aún, las funciones usadas en la línea 7 de la inicialización de la *Blockchain* y la línea 36 serán tratadas en profundidad en la sección 4.2.2, algoritmo 4.2.4 y en la sección 4.2.5, algoritmo 4.2.8 respectivamente, también serán incluidas en la clase.

En la implementación de la clase deben ser incluidos los algoritmos mostrados en el apartado ***Delegated Proof-of-Stake y Proof-of-Authority*** de la sección

3.6.2 Protocolos de consenso, además de que son necesarias funciones que utilicen los métodos GET y POST de HTML que permitan la interacción con los nodos autorizados, así como la conexión con los usuarios para recibir las transacciones. Por otro lado, una de estas funciones permitirá obtener la cadena y exportarla a un archivo JSON (explicado a detalle en la sección 4.2.6), de esta manera al finalizar el proceso, ésta puede hacerse pública y el conteo de los votos puede realizarse paralelamente al conteo oficial por cualquier persona o entidad.

**Algoritmo 4.2.1** Clase *Blockchain*

```
1:  Blockchain ( ):                                   ▷ Constructor de la clase
2:      cadena ← tablaHash();
3:      mempool ← [];
4:      lista ← lista();
5:      autoridades, direcciones ← set();
6:      No = bloqueAnterior ← 0;
7:      nodo ← direccionPrincipal();
8:      creaBloqueGénesis(No, bloqueAnterior, nodo);      ▷ Se crea el bloque génesis
9:
10: Función creaBloque(No, HashPadre, Autoridad, Peso):
11:     (Votos, Dx) ← separaVotos(self.mempool);
12:     Tm ← tiempo.ahora();
13:     MerkleRoot ← arbolMerkle(No, HashPadre, Tm, Peso, Autoridad, Votos, Dx);
14:     bloque(No, HashPadre, Tm, Peso, MerkleRoot, Autoridad, Votos, Dx);
15:     self.mempool ← [];
16:     self.cadena[MerkleRoot] ← bloque;
17:     self.bloqueAnterior ← MerkleRoot; self.lista.append(MerkleRoot);
18:
19: Función esValida(cadena):
20:     actual ← lista.inicio;
21:     sig ← actual.siguiente;
22:     Mientras sig ≠ null hacer:
23:         bloque ← cadena[sig.contenido];
24:         bloqueAnt ← cadena[actual];
25:         Si bloque['HashPadre'] ≠ arbolMerkle(bloqueAnt) entonces:
26:             regresa False;
27:         actual ← sig;
28:         sig ← actual.siguiente;
29:     regresa True;
30:
```

```
31: Función AGREGATRANSACCIÓN(remitente, voto, firma)
32:     tx ← {'Remitente':remitente,'Voto':voto, 'Firma':firma }
33:     Si Valida(tx) entonces:
34:         self.mempool.append(tx);
35: Función SEPARAVOTOS(mempool):
36:     Votos, Dx ← set();
37:     Para tx en mempool hacer:
38:         Dx.add(tx['Remitente'])
39:         direcciones.add(tx['Remitente'])
40:         Votos.add(tx['Voto'])
41:     regresa (Votos,Dx)
```

El algoritmo 4.2.2 será usado en las secciones 4.2.1 Etapa de Registro, y 4.2.3 Etapa de Autenticación del Usuario. Esta función se basa en los algoritmos 3.6.4, y 3.6.5; recibe una llave, un código cadena, y una tupla de índices sobre la que iterar, y devuelve una dirección anónima.

Dado que la variable *llave* es un número hexadecimal de 256 bits, la función *poly* de la línea 8, algoritmo 4.2.2 consiste en convertir esta variable a su representación binaria $B$, que tiene una longitud de 256 dígitos. Al convertir los dígitos en elementos de un arreglo, se obtiene de resultado la **representación en arreglo $p$, de un polinomio $P$**, cuyo grado máximo es 255.

Sea $U$ el número de dígitos 1 en $B$, si $U$ es par no se hace nada, de lo contrario, se agrega un "1" al final del arreglo, con lo que ahora el grado máximo de $P$ es 256, y $U+1$ es par. Dado que 257 es primo, en esta propuesta $N$ (requerido como parámetro del esquema NTRUEncrypt) queda definido como dicho número. Ahora, también requerido como parámetro, sea $d = \frac{U}{2}$ (o bien $\frac{U+1}{2}$ dependiendo de la paridad), entonces el polinomio $g$ requerido como una de las llaves privadas de la función $NTRU_{keygen}$ del algoritmo 3.4.1 es $p$ con los primeros $d$ dígitos "1" siendo convertidos a "-1". Los parámetros $f,p,q$ son predefinidos de antemano dados los requerimientos de la sección 3.4.

**Algoritmo 4.2.2** Dirección anónima en el árbol de direcciones, dados nivel e índices

1: **Función** ARBOLDIRNIVELI(*llave*, *cCadena*, *tupla*):
2: $tam \leftarrow \text{len}(tupla)$;
3: $i \leftarrow 0$;
4: **Mientras** $i < tam$ **hacer**
5: $(llave,\ cCadena) \leftarrow \text{llavesHijas}(llave,\ cCadena,\ tupla(i))$;
6: $i{+}{+}$;
7: $llavePública \leftarrow NTRU_{keygen}(poly(llave), (N, d, p, q))$;
8: $dir \leftarrow \text{anonDir}(llavePública,\ pre)$;
9: **regresa** *dir*;

### 4.2.1. Etapa de Registro

La Etapa de Registro se seguirá en la medida de lo posible como lo indique el protocolo de la institución encargada, sin embargo se implementará el uso de una aplicación para computadora que permita a la institución tener una base de datos del padrón de votantes en la que se cuente con información de estos, necesaria para esta propuesta.

Más específicamente, dado que es fundamental para esta propuesta que se cuente con un registro sobre alguna (o algunas) de sus huellas dactilares, una contraseña generada por el usuario al momento de su registro, (información que será guardada en forma de hash en la base de datos), esta aplicación se enfocará en obtener estos datos, agregarlos a la base de datos de la institución además de generar el código nemotécnico necesario para derivar una semilla que sirva para generar una HD wallet para uso del usuario al momento de las votaciones. Dicho código (o bien, la semilla) será almacenado en la tarjeta de identificación del usuario que certifique la pertenencia al padrón en forma de código de barras, código QR, o bien un chip NFC y almacenado en la base de datos de igual manera en forma de hash.

El pseudocódigo de dicha aplicación se muestra en el algoritmo 4.2.3, se usa la función Nemotécnica del algoritmo 3.6.3, y como *salt* para generar la semilla se utilizará la contraseña en texto plano otorgada por el usuario junto con la información

obtenida al escanear las huellas dactilares. La entropía es generada con una longitud de 256 bits mediante una combinación de movimientos aleatorios del ratón, la hora en la que se hace el registro (Unix time) y la ubicación (en caso de disponerla), o bien por entropía generada por el propio equipo físico.

Una vez realizado este proceso se utiliza el algoritmo 3.6.4 iterado 3 veces para crear un árbol de llaves profundidad 3, y los índices (0,0,0), es decir, el algoritmo 4.2.2 con la tupla (0,0,0) para generar una llave pública hija que servirá para crear una dirección anónima para el usuario (algoritmo 3.6.5) que será almacenada en la base de datos de la institución. Sea $U$ un usuario cualquiera, a partir de ahora nos referiremos a la dirección generada como **dirección pública de U** denotada como $PAd_U$. De la misma manera, se genera la llave de verificación del usuario (FDCE, sección 3.2), para almacenarlo junto con la $Pad_U$. Estos dos serán los únicos datos criptográficos que se registren y que permitan relacionarse con algún usuario en específico.

**Algoritmo 4.2.3** Programa para el registro de votantes

1: *pss* ← esperaContraseña();
2: *fngr* ← esperaHuella();
3: *entrp* ← obtenEntropía();
4: *código* ← nemotécnica(*entrp*);
5:
6: *salt* ← *pss*+*fngr* ;
7: *semilla* ← PKFD2(*código*,*salt*,HMAC-SHA512);
8:
9: *Pss* ← SHA256(*pss*);
10: *Code* ← SHA256(*código*);
11: *Principal* ← HMAC-SHA512(*semilla*);
12: *llave* ← *Principal*[0:255];
13: *cCadena* ← *Principal*[256:512];
14: $PAd_U$ ← arbolDirNivelI(*llavePública*, *cCadena*, (0,0,0));
15: $(Pu, PPu)$ ← generaLlaveFDCA$(g, p)$; ▷ Se generan las llaves del esquema FDCA con unos $g, p$ predefinidos
16: agregarDB(Usuario, *Pss*, *fngr*, *Code* $PAd_U$, $Pu, PPu$);

### 4.2.2. Etapa de Pre-votación

Esta etapa es principalmente administrativa y es realizada una vez comenzado el periodo electoral, puede llevarse a cabo días antes de la fecha de votaciones, o bien, el mismo día antes de iniciar la jornada electoral. Son necesarias tres cosas en esta fase. En primer lugar es necesario establecer un sistema de etiquetas para las opciones a elegir, junto con la etiqueta $NULL$ para el voto nulo, de modo que al ejercer el voto en la sección 4.2.4 Etapa de Votación, sea la etiqueta de la opción elegida la que es enviada. Este sistema de etiquetas será también distribuido en la sección 4.2.6Etapa de Conteo para quien desee realizar el conteo por su cuenta una vez finalizada la jornada electoral y así corroborar los resultados oficiales y el conteo realizado por la autoridad.

Como segunda parte, es necesario establecer un orden de minado entre los nodos autoridad (esto es, si $n$ es el número de nodos autoridad, se asigna $i \in \{0, ...n-1\}$ a cada nodo, siendo $p_i$ el nodo $i$, y $p_0$ el nodo principal del sistema), y establecer el *periodo* de minado, es decir, el tiempo que debe transcurrir entre la generación de un bloque y otro, el *periodo* será registrado en el bloque génesis en el apartado reservado para el *peso* del bloque. Estas configuraciones son necesarias para la etapa de Minado de los bloques, sección 4.2.5.

La última parte necesaria es inicializar la implementación de la clase *Blockchain* (algoritmo 4.2.1). Del orden establecido en el párrafo anterior, el nodo principal ($p_0$) será el encargado de firmar el bloque génesis. La función hace la petición de las direcciones a la base de datos de la lista nominal, y agrega una transacción a la mempool por cada dirección obtenida, haciendo una transferencia simbólica de un token de autorización, siendo el emisor el nodo principal predefinido ($p_0$).

Cada nodo es una instancia de la clase "*Blockchain*". Esta inicialización manda llamar la función "creaBloqueGénesis()" (algoritmo 4.2.4). Variación de la función "creaBloque()", esta función se encargará de registrar en la *Blockchain*, dentro del

bloque génesis, a los usuarios que están en la lista nominal de la institución, es decir aquellos usuarios que cumplen los todos los requisitos necesarios y suficientes para poder votar de modo que al momento de hacer pública la *Blockchain*, dicha lista también será transparente para su verificación al público. Esta lista nominal es creada por la institución y depende enteramente de protocolos de votación. Las direcciones de los usuarios serán obtenidas a partir de la base de datos generada en la etapa anterior siendo así la lista un subconjunto de la base de datos.

Únicamente dos cambios son realizados entre ambas funciones, el primero consiste en solicitar a la base de datos las direcciones que cumplan con los requisitos de la lista nominal, se agrega una transacción directamente a la mempool por cada dirección obtenida, sin embargo en el apartado de "Remitente" se introduce la dirección de Destino, de modo que en la lista $Dx$ del bloque génesis quede registrada la lista que nos interesa. El segundo cambio se encuentra en el apartado de *peso*, que para el bloque génesis guardará el dato *periodo*, este cambio no afecta en ningún modo al protocolo de consenso, dado que todos los nodos $p_i$ comparten el mismo bloque génesis, entonces al calcularse la diferencia de peso total entre cadenas, las variaciones se encontrarán en bloques posteriores.

**Algoritmo 4.2.4** Generación del Bloque Génesis

```
1:  Función creaBloqueGénesis(No, bloqueAnterior, nodo):
2:      Address ← requestBD("LISTA NOMINAL");
3:      Para cada address en Address hacer:
4:          self.mempol.append({address, nodo, NULL});
5:
6:      (Votos, Dx) ← separaVotos(self.mempool);
7:      Tm ← tiempo.ahora();
8:      periodo;                    ▷ Se modifica este dato dependiendo de la institución
9:      MerkleRoot ← arbolMerkle(No, HashPadre, Tm, periodo, nodo, Votos, Dx);
10:     bloque(No, HashPadre, Tm, periodo, MerkleRoot, nodo, Votos, Dx);
11:     self.mempool ← [ ];
12:     self.cadena[MerkleRoot] ← bloque;
13:     self.bloqueAnterior ← MerkleRoot;
14:     self.lista.append(MerkleRoot);
```

### 4.2.3. Etapa de Autenticación del Usuario

A lo largo de la jornada electoral, las personas registradas en la lista nominal procederán a emitir su voto de acuerdo a los protocolos de la institución, frente a la pantalla de la aplicación diseñada para esto. Como primera pantalla se encontraran con la que se muestra en la figura 4.1, presionando cada uno de los botones procederá a ingresar su contraseña, escanear su huella digital registrada y el código QR (o bien código de barras o chip NFC) de su identificación, mismos que registró en la Etapa de Registro (sección 4.2.1). Esto permite una autenticación del usuario en 3 pasos, algo que sabe, algo que es, y algo que posee respectivamente.

Entonces el programa hace la consulta a la base de datos con los datos del hash de la contraseña, el hash de la huella digital y el hash del contenido de la identificación del usuario para verificar la existencia de dicho usuario en el registro de la institución. En el caso de ser confirmada la identidad del usuario, la aplicación recrea la **dirección pública del usuario** $PAd_U$ (algoritmos 3.6.3, 3.6.4, 3.6.5 unidos en el algoritmo 4.2.2, como en el algortimo 4.2.3), y busca en el bloque génesis si se encuentra en la lista nominal y así pueda proceder a sufragar. El objetivo de recrearla y no solicitarla a la base de datos es evitar cualquier fuga de información que pueda poner en peligro la identidad de un usuario, o bien, la suplantación de identidad.

Acto seguido, se crea un árbol de firmas de nivel 6 con el algoritmo 4.2.2 usando como índices los últimos 6 dígitos del hash de la llave pública principal, raíz del Árbol Merkle, para generar una dirección anónima $AdV_U$ que será buscada en el conjunto *direcciones* de la *Blockchain*, para verificar si dicha dirección ya ejerció su voto.

Si todas estas condiciones se encuentran en regla, se mostrará en la pantalla la boleta para proceder a la Etapa de Votación (sección 4.2.4). En caso contrario, se mostrará una pantalla notificando al usuario que ya hizo uso de su voto efectivo.
(*) En el algoritmo siguiente se considera que es la semilla el dato almacenado en

la identificación, si es almacenado el código nemotécnico, la aplicación cambia para generar nuevamente la semilla a partir del código.

**Algoritmo 4.2.5** Autenticación del usuario

```
1: Función AUTH( ):
2:     pass ← esperaContraseña();
3:     fngr ← esperaHuella();
4:     datoID ← esperaCódigo();
5:
6:     Si requestDB(hash(pass), hash(fngr), hash(datoID) IN Users) entonces:
7:         semilla ← datoID;
8:         Si datoID es código nemotécnico entonces: ▷ Si pasa como en (*), de lo contrario se omite este paso
9:             salt ← pass+fngr;
10:            semilla ← PKDF2(datoID, salt, HMAC-SHA512);
11:        llaveP ← HMAC-SHA512(semilla)[0:255];
12:        llavePública ← NTRU_keygen(poly(llaveP), (N, d, p, q));
13:        PAd_U ← arbolDirNivel(llavePública,cCadena,(0,0,0));
14:
15:        Si PAd_U ∈ Blockchain.BloqueGénesis entonces:
16:            tupla ← llaveP[-6: ];
17:            AdV_U ← arbolDirNivelI(llavePública, cCadena, tupla);
18:            Si AdV_U ∈self. direcciones entonces
19:                PantallaVoto();                    ▷ Redirige a la pantalla de voto
                                                      ▷ Si el usuario ya votó, se redirige a otra pantalla
                                                      ▷ Si el usuario ya votó, se redirige a otra pantalla
                                                      ▷ Si el usuario no existe, se redirige a otra pantalla
```

### 4.2.4. Etapa de Votación

Una vez que el sistema verificó la identidad del usuario, además de que no ha votado, se mostrará la pantalla de opciones elegibles, como en una papeleta de votación tradicional. Las opciones aparecerán ordenadas de manera aleatoria de modo que no se pueda identificar la opción elegida por la posición en la que se encontraba, aquí el usuario emitirá su elección al seleccionar alguno de los recuadros.

Al haber tomado su decisión, la aplicación toma la etiqueta de la opción elegida, o bien la etiqueta $NULL$ en caso de que el usuario haya anulado su voto. Acto seguido se construye un archivo JSON como sigue:

$$\{"PAd_U" : PAd_U, "Firma_U" : M_U, "Voto" : C(H(E_{voto}))\} \tag{4.3}$$

Donde:

- $Pad_U$ es la dirección pública del usuario.
- $M_U$ es el hash de algún mensaje aleatorio predefinido, solo conocido por la autoridad y la aplicación de uso para los usuarios, firmado con el algoritmo de Firma Digital (ECDSA) del usuario.
- $C(H(E_{voto}))$ es el hash de la etiqueta elegida por el usuario (concatenada con un mensaje generado aleatoriamente), encriptada con la llave pública NTRU de la autoridad y cegada con el Esquema de Firma Digital Cegada.

Este archivo es enviado a la autoridad para que el voto sea devuelto firmado y así validado sin tenerse conocimiento del contenido del mismo. El procedimiento por parte de la autoridad es como sigue (4.2.7):

1. Recibido el archivo JSON, la autoridad busca en la base de datos la dirección $PAd_U$ y recupera la llave pública necesaria para comprobar la autenticidad de la firma. Si tiene éxito, busca la dirección $PAd_U$ en el conjunto *direcciones* de la *Blockchain*, para verificar que el usuario no intente que su voto sea firmado más de una vez, impidiendo un posible voto doble. En caso de no estar, se agrega la dirección a dicho conjunto y luego la desecha.

2. La autoridad verifica la firma con la llave pública, la firma $Firma_U$, y el mensaje que se puede recrear del mismo lado del servidor. Si tiene éxito, los anteriores elementos son desechados.

3. Si la firma es verificada y el usuario es válido, la autoridad procede a firmar el hash del mensaje cegado y encriptado $C(H(E_{voto}))$.

4. La autoridad regresa $S(C(H(E_{voto})))$ la firma del mensaje cegado.

5. En caso de fracasar en alguno de los puntos anteriores, se regresa un mensaje de error.

Una vez que el usuario tiene el mensaje firmado por la autoridad $S(C(H(E_{voto})))$, procede a descegarlo y así obtener el mensaje firmado por la autoridad y encriptado con la llave pública de la autoridad $S(H(E_{voto}))$. Acto seguido, junto con la dirección generada en la sección 4.2.3 $AdV_U$, se construye el archivo JSON que será enviado a validar y ser registrado en la *Blockchain*, siguiendo la estructura de 4.2:

$$\{\text{"}Remitente\text{"} : AdV_U, \text{"}Voto\text{"} : E_{voto}, \text{"}Firma\text{"} : S(H(E_{voto}))\} \tag{4.4}$$

Si todo ha procedido de manera correcta, se mostrará la pantalla de que el voto ha sido enviado con éxito y el usuario podrá retirarse. Después el voto es validado y se realiza el minado de los bloques, que se revisa en la siguiente sección.

**Algoritmo 4.2.6** Votación (Parte del Usuario)

```
1:  Función PantallaVoto( ):
2:      M_U ← F_sig(SHA(Mensaje), Priv); tag ← esperaTag(); K_aut ← llaveAut();
3:      E_voto ← NTRU_encrypt(tag, K_aut);
4:      C(H(E_voto)) ← cegar(H(E_voto))
5:      JSON = {"PAd_U" : PAd_U, "Firma_U" : M_U, "Voto" : C(H(E_voto))}
6:      S(C(H(E_voto))) ←envíaAfirma(JSON);
7:                              ▷ Se envía el archivo y se recibe el voto firmado y cegado
8:      S(H(E_voto)) ← decegar(S(CH(E_voto))
9:      tx = {"Remitente" : AdV_U, "Voto" : E_voto, "Firma" : S(H(E_voto))}
10:     enviarTransacción(tx); ▷ Al final, se le redirige a otra pantalla de conclusión.
```

**Algoritmo 4.2.7** Votación (Autoridad)

```
Función enviaAfirma(JSON):
    Si pub ← BD(JSON["PAd_U"]) && PAd_U ∉ direcciones entonces:
        Si Verify(JSON["Firma_U"], SHA(Mensaje), pub) entonces:
            self.direcciones.append(PAd_U);
            regresa firma(JSON["Voto"])
```

### 4.2.5. Etapa de Validación del Voto y Minado de los Bloques

Esta etapa consiste de dos partes: validar cada archivo JSON recibido para agregarlo a la *mempool* usando la función "agregaTransacción()" de la clase *Blockchain* (algoritmo 4.2.1) que manda llamar a la función "valida()" (algoritmo 4.2.8). Y el minado de los bloques que se realizará de la manera como se muestra en la sección 3.6.2, en el apartado ***Delegated Proof-of-Stake y Proof-of-Authority***.

Para la validación, se seguirá el procedimiento siguiente:

1. Una vez recibido el archivo JSON 4.4 se utiliza la función de verificación del Esquema FDC (sección 3.3), para verificar si el dato contenido en el apartado "$Firma$" del archivo coincide con la firma de la autoridad sobre el hash del dato en contenido en el apartado "$Voto$".

2. Si la firma es auténtica, devuelve True, y se procede a agregar la transacción a la *mempool*, de lo contrario, se rechaza la transacción y no es agregada.

**Algoritmo 4.2.8** Validación del voto

```
1: Función VALIDA(tx):
2:     E_voto ← tx["Voto"];
3:     S(H(E_voto)) ← tx ["Firma"];
4:     Si verifica(H(E_voto), S(H(E_voto))) entonces:
5:         regresa True;
6:     si no,
7:         regresa False;
```

Para el minado de los bloques, se seguirá el algoritmo Clique de la familia *Proof-of-Authority* (algortimo 3.6.2). Dado que los nodos autoridad se encuentran en un estado corriendo un bucle infinito, una vez transcurrido el *periodo* predefinido en la etapa de Pre-votación (sección 4.2.2), se toman turnos para mandar llamar la función "creaBloque" de la clase *Blockchain* (algoritmo 4.2.1), esta es la única modificación significativa que se muestra en el algoritmo 4.2.9.

**Algoritmo 4.2.9** Minado de los bloques (estado en el nodo $p_i$)

```
1: autoridades;                                  ▷ El conjunto de direcciones de los nodos autoridad
2: B_i ← Blockchain.cadena; P_i ← Blockchain.lista;
3: c_i ← ⟨B_i, P_i⟩;                             ▷ La copia de la Blockchain en el nodo i
4: periodo ← B_i[P_i[0]].peso;
5: mayoria ← ⌊|autoridades|/2⌋ + 1;
6: limite ← mayoria;
7:
8: Función FIRMADORECIENTE(c_i, n):
9:     λ ← limite;
10:    ret ← False;
11:    Para m=n-λ,...,n hacer:                   ▷ Iterar sobre los últimos λ bloques
12:        Si B_i[P_i[m]].no mod |autoridades| = i entonces:
13:            ret ← True
14:    regresa ret
15:
16: Función PROPONER( ):
17:    Mientras True hacer:
18:        n ← B_i[P_i[−1]].no;                  ▷ Índice del último bloque
19:        espera hasta que firmadoReciente(c_i, n) = False   ▷ Espera hasta que pueda firmar un bloque
20:        T ← B_i[P_i[n]].Tm;
21:        espera hasta tiempo.ahora() ≥ T + periodo
22:        Si (n + 1)mod |autoridades| = i entonces:   ▷ Turno del nodo p_i
23:            peso ← 2;
24:        si no,                                ▷ Sellado fuera de turno
25:            sleep(rand([0,500×mayoria]));     ▷ Retraso aleatorio en ms
26:            peso ← 1;
27:        Blockchain.creaBloque(n+1, B_i[P_i[0]].MerkleRoot, p_i, peso);
28:        distribuir();                         ▷ Se distribuye la copia de la cadena
29:
30: Función PESOTOTAL(⟨B_j, P_j⟩):
31:    regresa Σb.pesos : b ∈ B_j;
32:
33: Función ACTUALIZA(⟨B_j, P_j⟩):
34:    Si pesoTotal(⟨B_j, P_j⟩)> pesoTotal(⟨B_i, P_i⟩) entonces: ⟨B_i, P_i⟩ ← ⟨B_j, P_j⟩
```

A pesar de que en esta propuesta no se toma en cuenta, si el *periodo* es lo suficientemente corto, se puede usar la función "decidido()" para verificar que el bloque en el que el voto de un usuario ha sido *decidido*, y así notificarlo al mostrar la pantalla postvotación junto con el número de bloque.

### 4.2.6. Etapa de Conteo

Una vez finalizada la jornada electoral, se puede hacer pública la *Blockchain* como base de datos en un archivo JSON como se muestra en 4.5, de modo que cualquier persona pueda realizar el conteo de votos de manera paralela, o incluso tiempo después para verificar los resultados. Para realizar el conteo se sigue el algoritmo 4.2.10. El archivo JSON se compone de 3 elementos: la lista enlazada simple que sirve de índices para recorrer la cadena de bloques, la cadena compuesta por la tabla hash en la que cada objeto bloque de 4.1 puede ser recuperado a través de su $MerkleRoot$, y la lista con los nombres o bien etiquetas correspondientes a las opciones elegibles que fue establecida en la sección 4.2.2.

$$\{"Idx" : [MK_1, ..., MK_N], "Chn" : \{MK_1 : B_1, ..., MK_N : B_N\}, "Op" : [Op_1, ..., Op_M]\} \tag{4.5}$$

Con $N$ el número de bloques en la cadena y $M$ el número de candidatos elegibles. Si $Op$ es la lista de etiquetas, y para $i \in \{1, 2, ...M\}$, se definen $Op_i \in Op$ y $L_i$ como la lista de votos sobre la opción $i$. Así, se estructura el diccionario de listas de la manera siguiente:

$$\{Op_1 : L_1, ..., Op_M : L_M\} \tag{4.6}$$

En primer lugar, con ayuda de la lista de índices se itera la *Blockchain* en orden, ignorando el bloque génesis que no contiene información relevante para el conteo. Después, para cada objeto bloque recuperado, se extrae el conjunto de votos contenido dentro del bloque y se contabilizan mediante la ayuda del diccionario 4.6 al agregar cada voto a su correspondiente lista. El resultado final será el tamaño de cada lista.

**Algoritmo 4.2.10** Conteo de los votos

```
1:  Función CONTEO( ):
2:      json ← obtener.JSON();
3:      L ← json['Idx'];
4:      Lista ← L[1:];          ▷ Quitando el elemento correspondiente al bloque génesis
5:      Blockchain ← json['Chn'];
6:      Op ← json['Op'];
7:      dict ← dict();                                          ▷ Se crea el diccionario
8:      Para cada op en Op hacer:     ▷ Se inicializan las listas para los candidatos
9:          dict[op] ← [ ];
10:
11:     Para cada nodo en Lista hacer:
12:         bloque ← Blockchain[nodo];
13:         Votos ← bloque.votos;
14:         Para cada voto en Votos hacer:
15:             dict[voto].append(voto);
16:     Resultados ← dict();
17:     Para cada op en Op hacer:                ▷ Se toma el tamaño de cada lista
18:         Resultados[op] ← len(dict[op]);
19:
20:     regresa Resultados;
```

# Capítulo 5

# Conclusiones y trabajo a futuro

Como se planteó al inicio del Capítulo 4, se esperaba que la propuesta cumpliera 7 propiedades, las cuales son: **exactitud, privacidad, ausencia de recibo, robustez, verificabilidad, democracia y equidad.**

Podemos ver que el sistema es **exacto** ya que cada voto emitido es almacenado en la *Blockchain*, y que al ser liberada al público cualquier persona puede contrastar el conteo de los votos efectuado por sí mismo con los resultados oficiales. Incluso, el protocolo de consenso permite que aún habiendo una cantidad $n < \frac{|autoridades|}{2}$ de autoridades deshonestas el proceso seguirá siendo exacto.

Este sistema mantiene la **privacidad**, ya que la identidad de cada votante es escondida detrás de una serie de *direcciones públicas* generadas por una combinación de funciones hash criptográficas y la generación de llaves de un algoritmo criptográfico perteneciente actualmente a la familia de criptografía *postcuántica*, lo que hace prácticamente imposible obtener la dirección principal correspondiente a la identidad del votante, a partir de alguna de sus *direcciones públicas*. Además, en caso de que el usuario quiera encontrar una *dirección pública* concreta que lo identifique, es necesario que introduzca 3 elementos: una llave pública, un índice y un código cadena. Siendo los 2 primeros posiblemente públicos y accesibles para el usuario, no obstante el último permanece oculto para este, y es generado de manera local en la computadora que

realiza el cálculo, sin estar conectada a Internet. Con lo cual resulta imposible para cualquiera volver a generar la *dirección pública* deseada con la ausencia de dicho dato.

Dadas las razones mencionadas en el párrafo anterior, además del hecho de que en cada bloque de la *Blockchain* las *direcciones* y los votos son almacenados en conjuntos separados, sin que guarden relación alguna entre sí, podemos garantizar que el sistema cumple la propiedad de **ausencia de recibo**. Quizás el único punto en el que pueda encontrarse una debilidad que incumpla esta propiedad se encuentra en el momento en que la *dirección pública* y el voto se encuentran juntos; esto es, los archivos JSON de las ecuaciones 4.3 y 4.4. No obstante en el primer archivo (ecuación 4.3), el voto se encuentra difuminado por la función de cegado del Esquema de Firma Digital Ciega y debajo de la función de cegado se ocultó bajo un hash y encriptado por el algoritmo NTRUEncrypt con la llave pública de la autoridad, con lo que ninguna persona que tenga acceso a dicho archivo tendrá conocimiento del contenido del voto, a pesar de sí tener acceso a la dirección (la cual ya vimos que no identifica a ningún usuario).

Además, en el caso de que la función de cegado sea quebrantada, se tiene que romper también el hash y el encriptado con el cuál está oculto el voto, siendo de esta manera imposible de obtener el contenido del voto original, ya que al estar concatenado el voto con un mensaje aleatorio, es imposible recrear el mensaje intentando encriptar alguna de las etiquetas con la llave pública de la autoridad. El objetivo del hash es que en caso de que la función de cegado sea quebrantada, ni siquiera la autoridad pueda conocer el contenido del voto al no poder desencriptar el mensaje. De manera similar ocurre con el segundo archivo JSON (ecuación 4.4), que resulta imposible de recrear el mensaje intentando encriptar las etiquetas de votación.

Este sistema es **robusto** principalmente por la robustez de la *Blockchain*, esta robustez se obtiene principalmente por dos razones: la primera es la limitada capacidad de acción que tienen los usuarios que no son mineros, ya que las transacciones no son agregadas si no tienen un certificado que autentifique a un usuario honesto y no

pueden interactuar de otra manera con la estructura. Y la segunda es la propiedad del protocolo de consenso de tolerar un cierto número de mineros deshonestos sin afectar su correcto funcionamiento.
Por parte de ataques externos, dado que cada minero posee su propia copia de la estructura, para interrumpir su funcionamiento es necesario desconectar a todos los nodos autoridad, de este modo, mientras más nodos autoridad existan más difícil será que se interrumpa su funcionamiento.

El sistema es democrático porque permite que solo los usuarios registrados accedan a la pantalla de votación. En caso de que un atacante intente colar una transacción a la *Blockchain*, se encontrará con que su transacción debe estar firmada por la autoridad, sin embargo el proceso para obtener esta firma vuelve a involucrar la identificación del usuario registrado, de modo que un atacante que no se encuentre en el registro no podrá acceder a la firma, con lo cual su transacción no será verificada y no será agregada a ningún bloque. Por otro lado, la estructura *direcciones* auxiliar de la *Blockchain*, registra a los usuarios que ya votaron y a los usuarios que ya verificaron su voto, con lo que impide que quieran ejercer su voto más de una vez, no obstante, es independiente de los votos, con lo que no se puede relacionar ningún dirección ahí registrada con algún voto.

La **equidad** es cumplida al mantenerse en secreto la *Blockchain*, con lo que los votos y su conteo se mantiene desconocido para cualquier persona.

En cuanto a la **verificabilidad** se tienen 2 tipos: para la ***verificabilidad universal*** cualquier observador con el conocimiento necesario puede descargar el archivo JSON final de la *Blockchain* y realizar el conteo por sí mismo con lo cual puede convencerse del conteo final de la elección. Además, conociendo el proceso expuesto en este trabajo se puede convencer de la validez de los votos, pues como se expuso anteriormente, el sistema no permite votos que no sean validados correctamente.
Por parte de la ***verificabilidad individual*** el sistema tiene la posibilidad de indicar

al usuario el número de bloque en el que será almacenado su voto, sin embargo no puede comprobar que verdaderamente es su voto el que está almacenado ahí, pues entra en conflicto con otras propiedades mencionadas anteriormente. Dado que las direcciones que votaron se almacenan en el bloque en el que fue almacenado el voto, si se da a conocer la dirección final con la que el usuario votó, el usuario puede verificar que la dirección está en el bloque que se le indicó. No obstante si la cantidad de votos es reducida cabe la posibilidad de que se asocie una dirección con un voto, lo que acredita el contenido de un voto y se le relaciona con un votante, violando las condiciones de **privacidad** y **ausencia de recibo**.

**Trabajo futuro**

- Considerar cambiar el uso del SHA-1 por alguna función hash criptográfica mas segura, ya que al igual que el RIPEMD-160, también presenta vulnerabilidades. Se contempla la posibilidad de usar funciones hash criptográficas cuyas salidas sean de un tamaño mayor con motivo de mejorar la seguridad.

- Cambiar el algoritmo de consenso por alguno de la familia *Delegated Proof-of-Stake*, ya que el algoritmo Clique usado en esta propuesta es vulnerable a ataques del tipo *Attack of the clones*, que consiste en que una autoridad clona su par de llaves en dos instancias distintas que se comunican con distintos grupos de autoridades.

- Mejorar la aleatoreidad en la etapa de registro complementando el algoritmo usado con un algoritmo de números aleatorios mediante ruido atmosférico. Pues la aleatoreidad forma una parte fundamental en la seguridad de este sistema y se busca que la posibilidad de que dos personas generen el mismo número aleatorio sea mínima.

- Adicionar el uso de *smart contracts* para una implementación más inteligente del sistema y evitar el voto doble al programar la transacción inicial del bloque génesis de modo que al acceder cada usuario, solo pueda realizar una transacción, mejorando así la propiedad de **democracia**. Pues la solución a dicho

problema implica registrar una gran cantidad de información ya que por cada usuario que vota guarda dos direcciones, además puede presentar problemas

- Optimizar el proceso de conteo de modo que sea más rápido, ya sea mejorando el algoritmo o bien cambiarlo para que el proceso de conteo de varios bloques se pueda hacer de forma paralela.

- Mejorar la verificabilidad del sistema. Por parte de la verificabilidad universal, se trabajará en un mejor convencimiento de la validez de los votos individuales. Y por parte de la verificabilidad individual, se trabajará en un método en el que cualquier votante pueda verificar que su voto fue contado, sin entrar en conflicto con las demás propiedades mencionadas.

# Bibliografía


[1] European Computer Manufacturers Association (ECMA). *ECMAScript Language Specification 3rd Edition*. 1999.

[2] Andreas M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain*. O'Reilly, 2017, págs. 195-212. ISBN: 978-1-491-95438-6.

[3] Tim Bray. *RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format*. Internet Engineering Task Force (IETF), 2017.

[4] C. Cachin, R. Guerraoui y L. Rodrigues. *Introduction to reliable and secure distributed pro-gramming*. Springer, 2011. ISBN: 978-3-642-15260-3.

[5] Computer Security Research Center(CSRC). *Post-Quantum Cryptography Round 3 finalists*. National Institute of Standards y Technology (NIST), 2020.

[6] D. Chaum, P. Y. A. Ryan y S. Schneider. «A practical voter-verifiable election scheme». En: *Proceedings of the 10th European conference on Research in Computer security* ser. ESORICS'05 (2005), págs. 118-139. DOI: `http://dx.doi.org/10.1007/11555827_8`.

[7] David Chaum. «Blind signatures for untraceable payments». En: *Advances in Cryptology Proceedings of Crypto* 82 (1983), págs. 199-203.

[8] Cong Chen y col. *NTRU Algorithm Specifications and Supporting Documentation*. NTRU organization, 2019.

[9] Frank da Cruz. *Hollerith 1890 Census Tabulator*. URL: `http://www.columbia.edu/cu/computinghistory/census-tabulator.html`. (2019, 26 Diciembre).

[10] S. Davtyan y col. «Integrity of Electronic Voting Systems: Fallacious Use of Cryptography». En: *Symposium On Applied Computing (SAC 2012)* 27th (2012).

[11] Thomas A. Edison. «Electric Vote-Recorder». U.S. Patent 90,646. 1869.

[12] Parinya Ekparinya, Vincent Gramoli y Guillaume Jourjon. «The Attack of the Clones Against Proof-of-Authority». En: (2019). arXiv: `1902.10244v3 [cs.CR]`.

[13] S. Ezziri y O. Khadir. «A blind signature based on the DLP and RSA cryptosystem». En: *Gulf Journal of Mathematics* 6 Issue 4 (2018), págs. 44-50.

[14] Ariel J. Feldman, J. Alex Halderman y Edward W. Felten. «Security Analysis of the Diebold AccuVote-TS Voting Machine». En: *USENIX/ACCURATE Electronic Voting Technology Workshop* EVT'07 (2007).

[15] L. Fouard, M. Duclos y P. Lafourcade. «Survey on Electronic Voting Schemes». En: *VERIMAG 2* (2007).

[16] E. Gaetani y col. «Blockchain-based database to ensure data integrity in cloud computing environments». En: *ITA-SEC* 1816 (2017).

[17] Herman Hollerith. «The Electric Tabulating Machine». En: *Journal of the Royal Statistical Society* 57 (1894), págs. 678-682. DOI: `https://doi.org/10.2307/2979610`.

[18] A. Juels, D. Catalano y M. Jakobsson. «Coercion-resistant electronic elections». En: *Cryptoloy ePrint Archive* Report 2002/165 (2002). DOI: `http://dx.doi.org/10.1007/11555827`.

[19] Neal Koblitz. «Elliptic Curve Cryptosystems». En: *Mathemathics of Computation* 48 Number 177 (1987), págs. 203-209.

[20] H. Krawczyk, M. Bellare y R. Canetti. *RFC 2104: HMAC: Keyed-Hashing for Message Authentication*. Internet Engineering Task Force (IETF), 1997.

[21] S. Kremmer, M. Ryan y B. Smyth. «Election verifiability in electronic voting potocols». En: *Proceedings of the 15th European conference on Research in computer security* (2010), págs. 389-404.

[22] Leslie Lamport, Robert Shostak y Marshall Pease. «The Byzantine Generals Problem». En: *ACM Transactions on Programming Languages and Systems* (1982). DOI: https://doi.org/10.1145/357172.357176.

[23] Satoshi Nakamoto. «Bitcoin P2P e-cash paper». En: nov. de 2008. DOI: https://bitcoin.org/bitcoin.pdf.

[24] NIST. *Fips publication 180-2: Secure hash standard. Technical report.* National Institute of Standards y Technology (NIST), 2002.

[25] NIST. *Fips publication 180-4: Secure hash standard. Technical report.* National Institute of Standards y Technology (NIST), 2015.

[26] Marek Palatinus y col. *Mnemonic code for generating deterministic keys.* URL: https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki. (2013, 10 Septiembre).

[27] Sunil Philip y Abdus Samad Khan. *NTRU Python Library with Application to Encrypted Domain.* Columbia University, 2015.

[28] Stefan Popoveniuc y Benjamin Hosp. «An Introduction to PunchScan». En: ene. de 2010, págs. 242-259. DOI: 10.1007/978-3-642-12980-3_15.

[29] G. Z. Qadah y R. Taha. «Electronic voting systems: Requirements, design, and implementation». En: *Computer Standards & Interfaces* 29.3 (2007), págs. 376-386. DOI: https://www.sciencedirect.com/science/article/abs/pii/S0920548906000754.

[30] Michael Szydlo. «Merkle Tree Traversal in Log Space and Time». En: *Advances in Cryptology - EUROCRYPT 2004. EUROCRYPT 2004. Lecture Notes in Computer Science,* 3027 (2004). DOI: https://doi.org/10.1007/978-3-540-24676-3_32.

[31] K. Thompson. «Reflections on trusting trust». En: *Commun. ACM* 27.8 (1984), págs. 761-763. DOI: http://doi.acm.org/10.1145/358198.358210.

[32] Laura Vozzela. «Virginia scraps touch-screen voting machines as election for governor looms». En: *Washington Post* (2017). DOI: https://www.washingtonpost.com/local/virginia-politics/virginia-scraps-touch-screen-voting-machines-as-election-for-governor-looms/2017/09/08/e266ead6-94fe-11e7-89fa-bb822a46da5b_story.html.

[33] Frank S. Wood. «Electric voting-machine». U.S. Patent 616,174. 1898.