UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

Implementación de una blockchain resistente a ataques criptográficos cuánticos

Subtitulo del Proyecto

Autor
María Victoria Granados Pozo

Director
Gabriel Maciá Fernández
Francisco Javier Lobillo Borrero

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

FACULTAD DE CIENCIAS

—

Granada, noviembre de 2020

Implementación de una blockchain resistente a ataques criptográficos cuánticos

Subtítulo del proyecto.

Autor
María Victoria Granados Pozo

Director
Gabriel Maciá Fernández
Francisco Javier Lobillo Borrero

Granada, noviembre de 2020

Implementación de una blockchain resistente a ataques criptográficos cuánticos: Subtítulo del proyecto

María Victoria Granados Pozo

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3, ...

Resumen

Poner aquí el resumen.

Implementation of a blockchain resistant to quantum cryptographic attacks: Project Subtitle

María Victoria Granados Pozo

Keywords: Keyword1, Keyword2, Keyword3, ...

Abstract

Write here the abstract in English.

Yo, **María Victoria Granados Pozo**, alumno de la titulación Doble Grado de Ingeniería Informática y Matemáticas de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación y Facultad de Ciencias de la Universidad de Granada**, con DNI 77137043, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: María Victoria Granados Pozo

Granada a 7 de noviembre de 2020 .

D. **Gabriel Maciá Fernández**, Profesor del Área de Ingeniería Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

D. **Francisco Javier Lobillo Borrero**, Profesor del Área de Matemáticas del Departamento Álgebra de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Implementación de una blockchain resistente a ataques criptográficos cuánticos, Subtítulo del proyecto***, ha sido realizado bajo su supervisión por **María Victoria Granados Pozo**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 7 de noviembre de 2020 .

El director:

Gabriel Maciá Fernández **Francisco Javier Lobillo Borrero**

Agradecimientos

Poner aquí agradecimientos...

Índice general

Índice de figuras

Índice de tablas

Código

Capítulo 1

Introducción

El objetivo de este proyecto es evitar que un sistema *blockchain* sea vulnerable a futuros ataques cuánticos. Para ello se ha implementado un algoritmo criptográfico resistente a ordenadores cuánticos, denominado UOV, para la firma de documentos, y posteriormente integrarlo en la *blockchain* ARK.

1.1. Motivación y contexto del proyecto

La tecnología ha transformado nuestra sociedad en una sociedad digitalizada, donde actualmente, los dispositivos digitales comportan la mayor parte de nuestras actividades diarias en distintos ámbitos, como económicas, organizativas o sociales. En el proceso de digitalización de la sociedad podemos distinguir las siguientes cinco fases [5].

La primera fase o era del Internet, corresponde a mediados de los 90. En esta fase se comenzaron a crear páginas web para que los medios de comunicación y las empresas puedieran publicar y compartir información.

La segunda fase o era de las redes sociales, tuvo mayor auge a partir de 2005. Plataformas de bajo o ningún coste, se utilizaban en las empresas para poder llegar mejor a los clientes.

La tercera fase o era de la economía colaborativa, nació con la crisis de 2008 cuando las empresas tenían pocos recursos. Surgieron plataformas para conectar a las personas, y poder obtener lo que necesitasen unas de otras. Por ejemplo pagos online, ver recomendaciones y reseñas de un alojamiento o pedir un taxi. Además se da un gran paso ya que estas aplicaciones pasan de estar alojadas en ordenadores a teléfonos inteligentes.

La cuarta fase o era del mundo autónomo, se ha desarrollado durante décadas. Se desarrollan tecnología con inteligencia artificial, es decir, que simulan la inteligencia de los humanos para poder resolver problemas más complejos.

Quinta fase o era del bienestar moderno, comienza con las pulseras inteligentes como *Fitbit* o *Fuelband* de Nike. Estas pulseras son el impulso de la tecnología para facilitar la vida de los clientes y poder integrar la tecnología en la vida de los mismos.

La digitalización debe de venir acompañada de mecanismos que aporten seguridad a los datos. Los pilares de la seguridad de la información son los conocidos como la tríada CIA (confidencialidad, integridad y disponibilidad)[6].

La **confidencialidad** es la propiedad que impide que la información pueda ser accesible por entidades no autorizadas. Un sistema garantiza la confidencialidad cuando un tercero entra en posesión de la información intercambiada entre el remitente y el destinatario, no es capaz de extraer ningún contenido legible. Para asegurar la confidencialidad se utilizan mecanismos de cifrado y ocultación de la comunicación.

La **integridad** busca mantener la exactitud de los datos, es decir, que no hayan sido modificados durante su envío. La integridad se obtiene adjuntando al mensaje otro conjunto de datos de comprobación de la integridad, un ejemplo es la firma digital.

La **disponibilidad** es la cualidad de la información de encontrarse a disposición de quienes deben acceder a ella, ya sean personas, procesos o aplicaciones, en el momentos que así lo quieran. Los mecanismos para asegurar la disponibilidad se implementan con la infraestructura tecnológica.

Además de estos tres pilares hay otro principio, la **autentificación**, que es la propiedad que permite identificar al generador de la información. Trata de comprobar si un mensaje enviado por un usuario, ha sido verdaderamente firmado por él mismo. Esto se consigue con el uso de cuentas de usuario y contraseñas de acceso.

Para garantizar estos servicios de seguridad se hace uso de protocolos de seguridad de la información entre los que se encuentra la criptografía, la lógica y la autenticación.

La criptografía se ocupa de cifrar ciertos mensaje con el fin de hacerlos ilegibles a receptores no autorizados, una vez que llega a su destino y sea descifrado, el receptor obtendrá el mensaje original [7]. Además dota de seguridad a las comunicaciones, a la información y a las entidades que se comunican.

Podemos diferenciar dos tipos de criptografía, la criptografía simétrica y la asimétrica. La criptografía simétrica utiliza la misma clave para cifrar y descrifrar un mensaje, esta clave la ha de conocer tanto el emisor como el receptor. Mientras que la asimétrica utiliza dos claves la pública y la privada.

En la criptografía asimétrica podemos diferenciar dos ramas, el cifrado de clave pública y las firmas digitales [8]. En el cifrado de clave pública, el emisor cifra el mensaje con la clave pública del destinatario y el receptor lo descifra con su propia clave privada. En las firmas digitales, el emisor firma el mensaje con su clave privada y el receptor puede verificar el mensaje con su propia clave pública, además cualquier manipulación del mensaje se refleja en su resumen o *hash*.

Este tipo de criptografía basa su seguridad en la hipótesis de que no se pueden encontrar las claves por fuerza bruta con la tecnología existente en la actualidad. Los ataque de fuerza bruta tratan de recuperar las claves probando todas las posibles combinaciones hasta encontrar la que permite el acceso, a partir del algoritmo de cifrado y del texto cifrado con su original [9]. Para que la búsqueda tenga éxito se deberán de realizar $10^n - 1$ operaciones donde n es la longitud de la clave.

Otro factor importante, en la seguridad, es si en la clave aparencen números, caracteres o la combinación de ambos, aumentando así el coste de encontrar las claves, llegando a alcanzar tiempos de cálculo logarítmicos, es decir, que podrían tardar siglos en encontrar una contraseña compleja pero también depende de la capacidad de operación del ordenador.

En este contexto, la aparición de la futura computación cuántica permitirá el cálculo de operaciones a una velocidad mucho mayor. En la gráfica 1.1 podemos observar la capacidad de cómputo del peor ordenador cuántico, con la línea continua, que sigue la gráfica de una función exponencial, frente a la capacidad del mejor ordenador clásico, la línea discontinua, que sigue una función lineal.

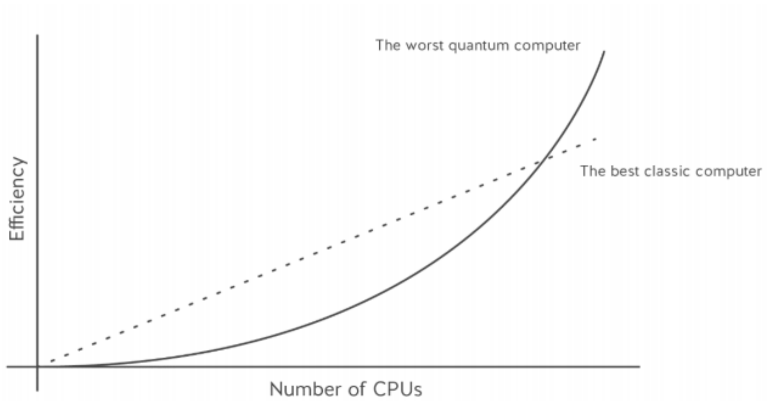

Figura 1.1: Comparativa de la capacidad de cómputo de un ordenador clásico con un ordenador cuántico[1]

La comparativa también nos muestra que para operaciones pequeñas como, por ejemplo, editar un documento de texto un ordenador cuántico sería probablemente ineficienciente. Por tanto lo mejor sería un ordenador híbrido, que mezclase computación clásica, para cálculos pequeños, y computación cuántica, para operaciones de mayor tamaño.

Cuando esté desarrollado el ordenador cuántico no serán válidos los actuales algoritmos criptográficos de clave pública, como RSA, Diffie-Hellman y ECDSA, ya que se basan en los problemas del logaritmo discreto y factorización de enteros, resolubles fácilmente por un ordenador cuántico. Las primeras ideas de la criptografía cuántica se tiene en los años 70, destacando los algoritmos de Shor y Grover.

Veamos la tabla comparativa 1.1, esta nos indica el tipo de algoritmo criptográfico, el algoritmo con la longitud de la clave y a continuación el nivel de seguridad tanto en un ordenador clásico como en uno cuántico. El nivel de seguridad de un algoritmo nos indica el número de operaciones necesarias para romper dicho algoritmo, por ejemplo, si tiene un nivel de seguridad n entonces se requieren 2^n operaciones para romper el algoritmo [10]. Observamos que hay una diferencia considerable en los niveles de seguridad de los algoritmo asimétricos, puesto que con un ordenador clásico al menos se necesitan 2^{112} operaciones mientras que con computación cuántica solo una.

Tipo	Algoritmo-Longitud clave	Nivel seguridad (ordenador clásico)	Nivel seguridad (ordenador cuántico)	Ataque cuántico
Asimétrico	RSA-2048	112	0	Algoritmo de Shor
	RSA-3072	128	0	Algoritmo de Shor
	ECC-521	128	0	Algoritmo de Shor
	ECC-521	256	0	Algoritmo de Shor
Simétrico	AES-128	128	64	Algoritmo de Grover
	AES-256	256	128	Algoritmo de Grover

Tabla 1.1: Niveles de seguridad de ordenadores clásicos y cuánticos [4]

En la actualidad, se están desarrollando muchos algoritmos para que sean resistentes a ataques de tipo cuántico, denominados algoritmos de criptografía postcuántica [11]. Estos ataques afectan principalmente a los algoritmos de clave pública o asimétrica, puesto que para la criptografía simétrica duplicar el tamaño de clave empleada es suficiente para hacerlos seguros y hacer inservible el algoritmo de Grover.

Por otro lado, también en la actualidad está siendo muy relevante la adopción de las *blockchain* como tecnología para ofrecer diversos servicios. Las *blockchain* o cadenas de bloques son listas de transacciones, denominadas bloques, firmadas y unidas con algoritmos criptográficos. Además cada bloque contiene el hash del bloque anterior, se explicará con más detalle en la sección 1.4.2.

Esta tecnología se ha integrado en diferentes áreas, donde resalta el uso en los servicios financieros o criptomonedas, que aumenta la eficiencia y disminuye los costes. Otro uso de las *blockchain* es en las cadenas de suministro, algunos restaurantes, como Fogo de Chão [12], están empezando a utilizar las *blockchain* para poder rastrear el origen de sus alimentos hasta llegar al propio restaurante, una gran ventaja para encontrar fácilmente si hay algún producto contaminado o en mal estado.

Un ejemplo del uso de las *blockchain* queda reflejado en este proyecto en el que se ha implementado un algoritmo resistente a ataques cuánticos, UOV [13] y se ha adaptado a la *blockchain* ARK [14] para que se utilice dicho algoritmo de firma.

1.2. Objetivos del proyecto y logros conseguidos

El objetivo de este proyecto es modificar el algoritmo de firma y verificación de las transacciones de la *blockchain* ARK, para hacerla resistente a ataques cuánticos.

- Implementación del algoritmo UOV: Se ha implementado las funciones de generación de claves tanto públicas como privadas, la función de firma a partir de la clave privada y la función de verificación de la misma con la clave pública. Además ha sido necesario implementar la aritmética de cuerpo finito de 2^7 elementos.

- Integrar el algoritmo UOV en la *blockchain* ARK para comprobar su funcionamiento: Se ha modificado el algoritmo de firma dado en la *blockchain* por el algoritmo UOV para aumentar la seguridad.

1.3. Estructura de la memoria

A continuación se muestran los capítulos que presenta la memoria junto con una breve descripción de lo que contiene cada uno.

1. Introducción: Presenta la motivación del proyecto y el contexto en el que surge, además incluye una breve reseña introduciendo las dos tecnologías que se han utilizado computación cuántica y la *blockchain*, aparte de la explicación matemática del algoritmo utilizado. En este capitulo también se encuentran los objetivos que se persiguen con este trabajo.

2. Planificación y costes: Contiene el diagrama de Gantt con la defición de las entregas y seguimiento del proyecto, así como el presupuesto del proyecto.

3. Análisis del problema: Descripción de las funcionalidades y requisitos, y análisis de los objetivos que se muestran en la sección 1.2.

4. Diseño: Se encuentra el diseño de la implementación del algoritmo UOV y el diseño del ecosistema ARK, donde se integrará el algoritmo de firma.

5. Implementación: Contiene la explicación de la implementación del algoritmo UOV y la aritemética del cuerpo finito de 2^7 elementos.

6. Evaluación y pruebas: Ejemplo de la firma de una transacción en el sistema ARK.

7. Conclusiones

1.4. Contenidos teóricos para la comprensión del proyecto

En los siguientes apartados se explica los contenidos claves de este proyecto, que son la computación cuántica 1.4.1, la tecnología *blockchain* 1.4.2 y el algoritmo UOV 1.4.3.

1.4.1. Computación cuántica

La evolución de la tecnología se ha basado principalmente en la reducción de los transistores para aumentar la velocidad, llegando a escalas de tan solo algunas decenas de nanómetros. Esto tiene un límite y es la eficiencia, puesto que al seguir disminuyendo el tamaño podrían dejar de funcionar correctamente. De ahí surge la necesidad de descubrir nuevas tecnologías, la computación cuántica [15]. Así la computación cuántica constituye un nuevo paradigma de la informática basado en los principios de la teoría cuántica.

Las tecnologías cuánticas nacieron del estudio de algunos fenómenos físicos que aún no se entendían bien, entre los años 1900 y 1930, dando lugar a una nueva teoría en la física, la Mecánica Cuántica. La Mecánica Cuántica es la rama de la física que estudia del mundo microscópico, los sistemas atómicos y subatómicos y su interacción con la radiación electromagnética [16].

Historia

La computación cuántica tuvo sus inicios en los años 50 cuando algunos físicos, como Richard Feynman, fueron pioneros en mencionar posibilidad de utilizar efectos cuánticos para realizar cálculos computacionales [15]. En la charla de Richard Feynman titulada “Simulación de la física con computadoras”, a principio de la década de los 80, expuso algunos cálculos complejos que se podrían realizar más rápido con un ordenador cuántico.

A finales de los años 60, Stephen Wiesner escribe un artículo titulado “Conjuate Coding”, donde expone un primer acercamiento a la criptografía cuántica, el artículo fue publicado en los años 80 [17].

En 1981 Paul Benioff expone las ideas esenciales de la computación cuántica acompañada de su teoría, en la que propuso que un ordenador clásico trabajara con algunos pricipios de la mecánica cuántica, y aprovechar así las leyes cuánticas.

En la década de 1990 ya empezaron a poner en práctica algunas teorías, apareciendo los primeros algoritmo cuántidos, primeras aplicacines cuánticas y las primeras máquinas diseñadas para realizar cálculos cuánticos. Así en 1991, Artur Ekert desarrolla una aproximación diferente a la distribución de claves cuántica (QKD) basado en el entrelazamiento cuántico.

En 1993 hubo varios acontecimientos, por un lado, Dan Simon comparó el modelo de probabilidad clásica con el cuántico, esto se utilizó para el desarrollo de futuros algoritmos cuánticos. Por otro, Charles Benett acuñó el término del teletransporte cuántico, abriendo una vía de investigación para las comunicaciones cuánticas. Además Ekert organizó la primera conferencia internacional de criptografía cuántica en Inglaterra, primer evento de gran alcance dedicado a este área.

Peter W. Shor definió un algoritmo cuántico, el algoritmo de Shor, que permite calcular los factores primos de números muy grandes en tiempo polinimial, resolviendo el problema de la factorización de enteros como el problema del logaritmo discreto. Como consecuencia, el algoritmo de Shor permite romper muchos sitemas criptográficos actuales. Un año más tarde, propuso un sistema de corrección de errores en el cálculo cuántico.

Lov Grover, en 1996, expone un algoritmo de búsqueda en una secuencia de datos no ordenada con N elementos, denominado algoritmo de Grover, que tiene una complejidad en tiempo de $O(\sqrt{n})$.

En 1997, tiene lugar el primer experiemento de comunicación con criptografía cuántica a una distancia de 23km. Además del primer teletransporte cuántico de un fotón.

A finales de los 90, los laboratorios IBM-Almaden crearon la primera máquina con 3 cúbits y ejecutó el algoritmo de Grover. Y en 2001, IBM junto con la Universidad de Stanford ejecutaron el algoritmo de Shor en un computador cuántico con 7 cúbits, se calcularon los factores primos de 15.

En 2004, sale a la luz el primer criptosistema cuántico comercial(QKD), creado por la ID Quantique.

En 2007, D-Wave fabricó una máquina que utilizaba mecánica cuántica con 16 cúbits sin llegar a ser un computador cuántico, especializado en la optimización de problemas a través de algoritmo de temple cuántico. En septiembre de ese mismo año, consiguieron unir componentes cuánticos a través de superconductores, apareciendo el primer bus cuántico capaz de reter información cuántica durante un corte espacio de tiempo antes de volver a ser transferida. Un año después se consiguió almacenar un cúbit en el interior del núcleo de un átomo de fósforo y hacer que la información permaneciera intacta durante 1.75

segundos.

Pasaron varios años hasta que se vendió la primera computadora cuántica comercial, en 2011, por la empresa D-Wave Systems por 10 millones de dólares.

En 2018, la Universidad de Innsbruck consiguen un entrelazamiento estable de 20 cúbits, marcando el récord actual. El 18 de septiembre de 2019, IBM anunción que pronto lanzará un ordenador cuántico de 53 cúbits, el más grande y potente hasta la fecha.

En la actualidad, Google ha logrado aplicar una supercomputadores al mundo real, simulando con éxito una reacción química simple. Marcando el camino hacia la química cuántica. Esto podría ayudar a los científicos a comprender mejor las reacciones moleculares, dando lugar a descubrimientos útiles como mejores baterías, nuevas formas de producir fertilizante y métodos para eliminar el dióxido de carbono del aire [18].

Estructura de los cúbits

La computación clásica funciona con bits cuyos valores pueden ser 0 o 1, mientras que la computación cuántica funciona con bit cuánticos o cúbits, una combinación de 0 y 1, pudiendo tomar ambos valores a la vez, esto se denomina la superposición cuántica de los estados [19], se entrará en detalle más adelante.

La figura 1.2 muestra los estados de un bit y los posibles estados que puede tomar un cúbit.

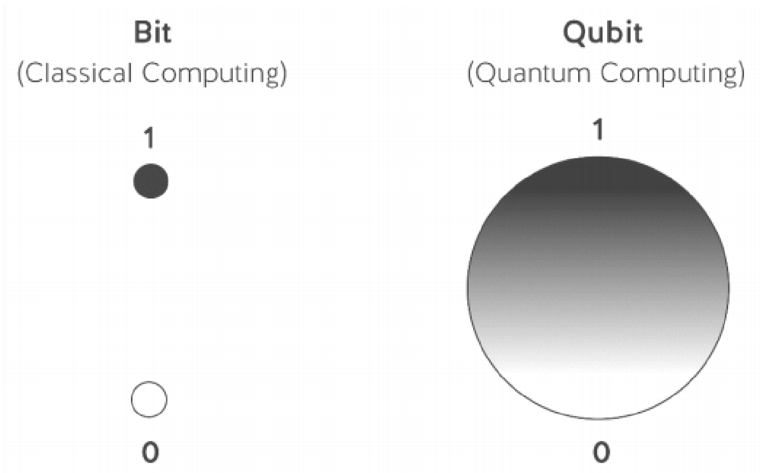

Figura 1.2: Estados de un bit y de cúbit [1]

El espacio de estados de un cúbit se puede representar mediante un espacio vectorial complejo bidimensional, al no ser práctico, se aprovecha el homeomorfismo entre la superficie de una esfera y el plano complejo cerrado con un punto en el infinito, dando lugar a lo que se conoce como la esfera de Bloch.

Una esfera de Bloch es una representación geométrica del espacio de estados puros de un sistema cuántico de dos niveles. Además se representa en el espacio $\mathbb{R}^3$ por la esfera de radio unidad como se observa en la imagen 1.3, donde cada punto de la esfera es un posible estado del cúbit.

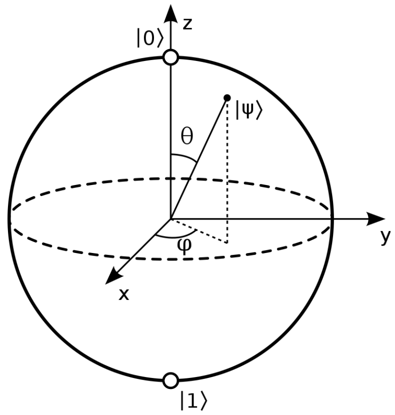

Figura 1.3: Estructura cúbit, esfera de Bloch [2]

Un cúbit se puede representar como una combinación lineal de los estados $|0\rangle$ y $|1\rangle$, ecuación 1.1.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.1}$$

Como α y β son números complejos, la ecuación 1.1 se puede escribir en forma exponencial, ecuación 1.2.

$$|\psi\rangle = r_\alpha e^{i\phi_\alpha}|0\rangle + r_\beta e^{i\phi_\beta}|1\rangle \tag{1.2}$$

Propiedades

Entre las propiedades cuánticas destacan la superposición cuántica, el entrelazamiento cuántico y el teletransporte cuántico.

La **superposición cuántica** describe cómo una partícula puede estar en diferentes estados al mismo tiempo. Esto aporta gran capacidad de procesamiento, lo que hace posible resolver de manera eficiente problemas de mayor complejidad como la factorización de enteros, el algoritmo discreto y la simulación cuántica, que a día de hoy con los ordenadores clásicos son difíciles de romper.

Otro aspecto importante de la física cuántica relacionado con la superposición es el **entrelazamiento cuántico** de las partículas[20]. Esto es, si dos partículas en algún instante han interactuado retienen un tipo de conexión y pueden entrelazarse formando pares, de forma que al interactuar con una de las particulas, por muy separadas que estén, la otra se entera. Esto permite que aunque los cúbits estén separados interactúen entre sí. Con estos dos aspectos la capacidad de procesamiento aumenta considerablemente, cuántos más cúbits la capacidad de procesamiento aumenta considerablemente.

Por último, el **teletransporte cuántico** utiliza el entrelazamiento para enviar información de un lugar del espacio a otro sin necesidad de viajar a través de él.

1.4.2. Blockchain

Blockchain es un sistemas de almacenamiento de información que se divide en bloque de datos enlazados mediante los hash. A cada bloque se le asocia un hash a partir del bloque anterior, creando una lista enlazada, la búsqueda de información no es muy óptima si hay un número elevado de bloques. Para la búsqueda eficiente en *blockchain* se usan los árboles merkle.

Los datos que almacena cada bloque son transacciones válidas, información referente a ese bloque y la relación con el bloque anterior mediante el *hash*, por tanto el bloque tiene un lugar específico dentro de la cadena. De esta forma si hay una alteración en un determinado bloque se verá reflejado en su *hash* y en el de los bloques posteriores, haciendo que la información de la cadena no se pueda perder, modificar o eliminar.

Los árboles merkle [21] son una estructura de datos en árbol en el que cada nodo que no es hoja está etiquetado con el *hash* que surge de la combinación de los valores o etiquetas de sus nodos hijo. Esta estructura permite que aunque los datos estén separados puedan ser ligados a un único valor de *hash*, el *hash* del nodo raíz del árbol. El *hash* de este nodo va firmado para asegurar la integridad y hacer que la verificación sea fiable.

De esta forma se asegura que los datos son recibidos sin daños y sin ser alterados, además permite que los datos puedan ser entregados por partes, ya que un nodo puede obtener solo la cabecera de un bloque desde una fuente y otra pequeña parte del árbol desde otra fuente, y poder asegurar que los datos son correctos. Esto funciona porque si un usuario intenta hacer un cambio en una transacción falsa en la parte inferior del árbol en seguida se verá reflejado en la parte superior del árbol, es decir, en el nodo raíz. Esta propiedad se ve clara en la estructura del árbol Merkle 1.4, donde los *hash* de los nodos superiores se calculan a partir de los *hash* de los nodos hijos.

La idea de la tecnología *blockchain* surge a comienzos de 1991 cuando los científicos Stuart Haber y W. Scott Stornetta introducen una solución computacional para la firma de documentos digitales y que no puedieran ser modificados con el tiempo. Usaron cadenas de bloque para almecenar los documentos con sello de tiempo y en 1992 se incorporaron los árboles Merkle, que podían recopilar varios documentos en un bloque haciendo el diseño más eficiente. Sin embargo, esta tecnología no se utilizó y la patente caducó en 2004 [22].

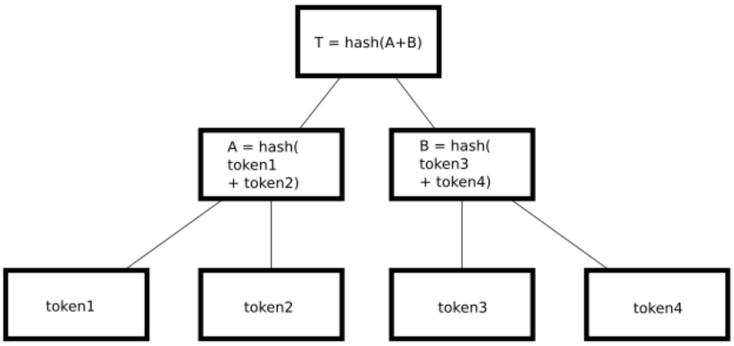

Figura 1.4: Estructura de un árbol Merkle [3]

En 1998, Nick Szabo trabaja en una moneda digital descentralizada, “bit gold”. Dos años después Stefan Konst publica su teoría sobre la seguridad criptográfica en las cadenas de bloques junto con algunas ideas de implementación [23].

En 2004, el informático y criptógrafo Harold Thomas Finney introdujo el sistema RPoW (prueba de trabajo reutilizable). El sistema se basa en *HashCash* pero los token de prueba no están ligados a una aplicación sino que pueden ser gastados libremente como una moneda. Los clientes pueden crear tokens e intercambiarlos sin necesidad de regenerarlos [24]. RPoW resolvió el problema del doble gasto registrando los tokens en un servidor fiable diseñado para permitir a los usuarios verificar su exactitud e integridad en tiempo real. Este sistema puede considerarse como un prototipo de las criptomonedas.

A finales de 2008, un grupo de desarrolladores bajo el nombre de Satoshi Nakamoto publican un documento técnico en que se establece un modelo para *blockchain*. Está basado en el algoritmo RPoW pero en lugar de usar dicho hardware, se utiliza un protocolo descentralizado peer-to-peer para verificar y restrear las transacciones. En otras palablas los “mineros” extraen bitcoins para obtener una recompensa mediantes pruebas de trabajo y posteriormente los nodos los verifican. Bitcoin nació el 3 de enero de 2009 cuando Satoshi Nakamoto extrajo el primer bloque de bitcoin con una recompensa de 50 bitcoins. Y el 12 de enero de 2009 tuvo lugar la primera transacción entre Satoshi Nakamoto y Hal Finney que obtuvo 10 bitcoins.

A partir de 2014, se comienzan a explorar el potencial de las cadenas de bloque y a buscar otras aplicaciones fuera de su uso en las transacciones financieras. Ethereum introduce programas informáticos que se ejecutan en la *blockchain*, se pueden utilizar para realizar una transacción cumpliendo ciertas condiciones como los contratos inteligentes.

Un contrato inteligenete se tratan de contratos que tienen la capacidad de cumplirse de forma automática. Un contrato inteligente está constituido por un protocolo de códigos que permiten a un dispositivo ejecutar de forma automatizada las sentencias previamente programadas, prescindiendo de la intervención humana [25].

Además de los contratos inteligentes, Ethereum tiene su propia criptomoneda llamada Ether, se puede transferir entre cuentas y se utiliza para pagar las tarifas por la ejecución de los contratos inteligentes.

Actualmente las *blockchain* tiene otros usos más allá de las criptomonedas.

Las cadenas de bloques o *blockchain* permiten verificar, validar, rastrear todo tipo de información, ya sean contratos inteligentes, transacciones financieras, certificados digitales o firmas [26], siendo estas últimas el centro de este trabajo. También permiten impulsar modificaciones orientadas a crear soluciones más robustas, por ejemplo en centros de salud o notarías que se explicarán más adelante.

Las *blockchain* son vulnerables a futuros ataques cuánticos ya que su única línea de defensa es el algoritmo de firma de los bloques. Aunque, actualmente las cadenas de bloques son seguras, puesto que un ordenador clásico no tiene la capacidad de cómputo necesaria para descifrar cada bloque, obtener la información y volver a firmar todos los bloques sin dejar huella. Por eso para hacer una *blockchain* resistente es necesario tener un criptosistema que no se pueda romper con computación cuántica, como por ejemplo el algoritmo UOV, ver apartado 1.4.3.

Los tres pilares de la tecnología *blockchain* son la descentralización, transparencia e inmutabilidad [27].

Un sistema centralizado almacena todos los datos en una misma entidad y habría que interactuar con la misma para obtener la información necesaria. Un ejemplo de un sistema centralizado son los bancos que almacenan todo el dinero y la única forma de pagar a alguien es a través de un banco. Es similar a la arquitectura cliente-servidor donde los clientes se comunican entre ellos mediante el servidor. Pero tener un único sitio para almacenar todos los datos es vulnerable a los ataques, informáticos, por otra parte si el nodo central se corrompe o tiene una actualización, los datos será incorrectos o no se podrán acceder a ellos. De los contras de los sistemas centralizados surge la idea de los sistemas **descentralizados**, la información no la tiene un único nodo sino que todos los usuarios son dueños de la información. La principal ideología de las *blockchain* es poder interactuar usuario con usuario sin tener que pasar por un tercero.

El concepto **transparencia** se refiere a la transparencia de los datos no de las identidades. Esto es la identidad de la persona se oculta a través de la criptografía y lo único que se ve es su dirección pública, pero podemos ver todas las transacciones que se han realizado en su dirección pública. En el historial de transacciones no vemos “Antonio envió 1BTC” sino que aparece “1MF1bhsFLkBzzz9vpFYEmvwT2TbyCt7NZJ envió un 1BTC”. Este nivel de transparencia nunca antes había existido en el sistema financiero, lo que exige más responsabilidad a las grandes empresas. De la misma forma podemos trasladar este concepto fuera del sistema financiero por ejemplo a las cadenas de suministro, y saber exactamente de donde provienen los alimentos de un restaurante.

La **inmutabilidad** en el contexto de las cadenas de bloques significa que una vez introducida una transacción en la *blockchain* ya no se puede alterar. De esta forma aplicando esta tecnología a los bancos se evitarían casos de malversación de fondos. Esta propiedad se obtiene gracias a la función criptográfica *hash*.

La función *hash* es el resultado de aplicar una función que transforma un mensaje de longitud variable en uno de longitud fija. Esto es calcular el resto módulo n con n la longitud fija. Al aplicar la función hash a un fichero, si se modifica algún dato del mismo cambiará su hash y por tanto se sabrá si ha sido manipulado desde que se envió, consiguiendo la integridad del mensaje.

De la misma forma si hay un cambio en una de las transacciones de un bloque se reflejará en el *hash* del bloque, afectado a todos los bloques anteriores. Así si el atacante quiere preservar la integridad deberá de modificar todos los bloques siendo una tarea imposible. De esta forma se obtiene la inmutabilidad de los datos.

Hoy en día, la tecnología *blockchain* está ganando mucha atención, no limitandose solo al uso en las criptomonedas. Así las cadenas de bloques tienen diversas aplicaciones entre ellas se encuentra la salud o la firma de documentos en las notarías. En el primer caso, cada centro de salud podría tener el historial médico de cualquier paciente, de una forma segura y evitando falsificaciones, estos historiales se encontrarían en nodos distribuidos de forma descentralizada así se obtendría un acceso rápido y seguro. El segundo caso será en el que nos centraremos a lo largo de este proyecto. Hoy día la firma de documentos o transacciones por parte de un usuario es un problema puesto que se pueden copiar con facilidad, pero con *blockchain* no podrían ser falsificadas debido a la propiedad de validación y rastreo de los datos.

1.4.3. Algoritmo UOV

El algoritmo aceite y vinagre desequilibrado es una versión simplificada del algoritmo aceite y vinagre, ambos algoritmos de firma digital. Para crear las firmas y validarlas es necesario resolver un sistema con m ecuaciones y n variables, que es un problema NP-duro, lo que significa que si fuésemos capaces de resolverlo con un ordenador cuántico, todos los problemas considerados en la actualidad serían vulnerables. Si m y n son casi iguales o iguales será más difícil resolver el sistema, obteniendo de esta forma un algoritmo de firma resistente a ataques cuánticos.

La principal ventaja del algoritmo UOV es que es un algoritmo post-cuántico, a diferencia de otros esquemas de firmas como RSA, DSA, basado en el logaritmo discreto, y su variante para curvas elípticas ECDSA que no permanecerían seguros ante un ordenador cuántico. Esto se debe a que en la actualidad no existe un algoritmo eficiente para la resolución de sistemas multivariados de ecuaciones en ordenador cuántico. Otra ventaja es la simplicidad de las operaciones utilizadas, ya que las firmas se crean y validan con operaciones de suma y multiplicaciones de valores pequeños, lo que requiere bajos recursos *hardware*.

Aunque el algoritmo usa sistemas pequeños y la longitud de las firmas son pequeñas, se necesitan claves públicas bastante más grandes en comparación con otros algoritmos de firma como ECDSA, pudiendo ocupar dicha clave pública varios kilobytes de almacenamieno. Por otro lado ya se conocen algunos métodos de ataque, probablemente aparecerán más si se empiza a comercializar.

Cuerpos finitos

Se trabajará con el cuerpo finito de 128 elementos, $\mathrm{GF}(2^7)$, extensión de grado 7 del cuerpo $\mathrm{GF}(2)$ de los enteros módulo 2

$$\mathrm{GF}(128) = \frac{\mathrm{GF}(2)[x]}{\langle x^7 + x + 1 \rangle} \tag{1.3}$$

Además el orden del cuerpo de las unidades es 127, que es primo entonces todo elemento del cuerpo distinto de 1 es un elemento primitivo, es decir, un generador.

La tabla 1.2 muestra una representación de los elementos no nulos del cuerpo. En la implementación se ha utilizado la representación como cadena de bits, puesto que a la hora de trabajar es más fácil con una cadeba de bits que con los polinomios.

La implementación del cuerpo finito de 2^7 elementos no se ha realizado de forma genérica sino para que sea específica para el algoritmo UOV, de esta forma es mucho más sencillo implementar la aritmética del cuerpo. Para la suma en $\mathrm{GF}(2)$ sólo tenemos que fijarnos que es lo mismo que el operador lógico *XOR*, mientras que para el producto, al encontrarmos en un cuerpo como un orden pequeño, se usarán unas tablas que contienen las correspondencias entre los elementos no nulos del cuerpo y sus logaritmos en base a, por lo que el producto se convierte en una suma módulo 127.

Parámetros y fórmula

Para empezar indicamos los parámetros que serán de utilidad para entender el algoritmo.

Polinomio	**Bits**	$\log_a$
1	[0, 0, 0, 0, 0, 0, 1]	0
a	[0, 0, 0, 0, 0, 1, 0]	1
a^2	[0, 0, 0, 0, 1, 0, 0]	2
$\vdots$	$\vdots$	$\vdots$
$a^6 + a^5 + a^4 + 1$	[1, 1, 1, 0, 0, 0, 1]	124
$a^6 + a^5 + 1$	[1, 1, 0, 0, 0, 0, 1]	125
$a^6 + 1$	[1, 0, 0, 0, 0, 0, 1]	126

Tabla 1.2: Representación de los elementos no nulos de GF(128)

- r: Grado del cuerpo extendido, $\mathbb{F}_2 \subset \mathbb{F}_{2^r}$. En la implementación se va tomar r igual a 7, pero se puede realizar con cualquier valor de r sin un esfuerzo adicional.
- m: Tamaño de la clave pública, además del número de variables de aceite.
- v: Número de variables vinagre.
- n: Número total de variables, las de aceite más las de vinagre.
- x: Vector de n componentes, denominando a las primeras v componentes $x_1, \cdots, x_v$ vinagre y al resto aceites.

$\mathscr{P} : \mathbb{F}_{2^r}^n \to \mathbb{F}_{2^r}^m$, esta función se puede descomponer como $\mathscr{P} = \mathscr{F} \circ \mathscr{T}$, donde $\mathscr{T} : \mathbb{F}_{2^r}^n \to \mathbb{F}_{2^r}^n$ es invertible, y $\mathscr{F} : \mathbb{F}_{2^r}^n \to \mathbb{F}_{2^r}^m$ siendo sus m componentes de la forma:

$$f_k(x) = \sum_{i=1}^{v} \sum_{j=i}^{n} \alpha_{i,j,k} x_i x_j + \sum_{i=1}^{n} \beta_{i,k} x_i \tag{1.4}$$

donde $\alpha_{i,j,k}$ y $\beta_{i,k}$ se toman aleatoriamente en $\mathbb{F}_2$ siendo α un vector de matrices triangulares superiores. De esta manera será más eficiente y no afectará a la seguridad del algoritmo.

Generación de la clave privada

La clave privada está formada por $\alpha_{i,j,k}$ y $\beta_{i,k}$ que son valores de GF(2), elegidos de forma aleatoria.

Generación de la clave pública

Generaremos una clave pública partiendo de la clave privada $\alpha_{i,j,k}$ y $\beta_{i,k}$.

Para entenderlo mejor ponemos las m ecuaciones (1.4) en forma matricial, ecuación 1.5. La notación a seguir para las matrices transpuestas es $(X)'$, con X una matrix.

$$f_k(x) = x^v \; [\alpha_{i,j,k}] \; (x^v, x^m)' + [\beta_{i,k}] \; (x^v, x^m)' \tag{1.5}$$

siendo $[\alpha_{i,j,k}]$ y $[\beta_{i,k}]$ las representaciones matriciales de $\alpha_{i,j,k}$ y $\beta_{i,k}$, x^v los vinagres y x^m los aceites, así x se puede expresar como $(x^v, x^m)'$.

Conociendo los valores de la clave privada α y β, tomando de forma aleatoria los del vinagre x^v, los cuales pasaremos a denominarlos como a^v, y cogiendo los m primeros bits del hash del mensaje h_k podemos generar la clave pública.

Hacemos el cambio de notación $A_k = a^v \; [\alpha_{i,j,k}] = (A_k^v, A_k^m)$, lo sustituimos en la ecuación (1.5) y despejamos los aceites.

$$h_k = A_k^v \; (a^v)' + A_k^m \; (x^m)' + \beta_k^v \; (a^v)' + \beta_k^m \; (x^m)' \tag{1.6}$$

$$(A_k^m + \beta_k^m)(x^m)' = h_k - (A_k^v + \beta_k^v)(a^v)' \tag{1.7}$$

$$(x^m)' = (A_k^m + \beta_k^m)^{-1}(h_k - (A_k^v + \beta_k^v)(a^v)') \tag{1.8}$$

Si $(A_k^m + \beta_k^m)$ fuese una matriz singular, entonces se tomarían otros valores de vinagres.

Para generar la clave pública necesitamos incluir una nueva matriz T, donde $T \cdot s' = x'$. Incluimos esta matriz T para aumentar la seguridad del algoritmo y así sea más complejo calcular la función inversa $\mathscr{P}$

$$T = \left[\begin{array}{c|c} I_v & T_{vxm} \\ \hline 0 & I_m \end{array}\right] \tag{1.9}$$

Despejando x, obtenemos:

$$x = s \cdot T' = s \left[\begin{array}{c|c} I_v & 0 \\ \hline T'_{vxm} & I_m \end{array}\right] = [s^v, s^m] \left[\begin{array}{c|c} I_v & 0 \\ \hline T'_{vxm} & I_m \end{array}\right] = (s^v + s^m T'_{vxm}, s^m) \tag{1.10}$$

Sustituimos en (1.5):

$$f_k(x) = s \left[\begin{array}{c} I_v \\ \hline T'_{vxm} \end{array}\right] [\alpha_{i,j,k}]_{\substack{1 \leqslant i \leqslant v \\ i \leqslant j \leqslant n}} \; T \; s' + [\beta_{j,k}]_{1 \leqslant j \leqslant n} \; T \; s' \tag{1.11}$$

donde $k \in \{1, \ldots, m\}$

Así obtenemos las claves públicas definidas para cada k

- $\alpha_{pub_k} = \left[\begin{array}{c} I_v \\ \hline T'_{vxm} \end{array}\right] [\alpha_{i,j,k}]_{\substack{1 \leqslant i \leqslant v \\ i \leqslant j \leqslant n}} \; T$
- $\beta_{pub_k} = [\beta_{j,k}]_{1 \leqslant j \leqslant n} \; T$

Algoritmo de firma

Gracias a la definición dada de x, en la ecuación 1.10, podemos hacer un simple despeje y calcular s. Esto se debe a que el sistema de ecuaciones se vuelve lineal cuando las variables de vinagre son fijas y además ninguna variable de aceite se multiplica por otra de aceite en la ecuación. Por tanto se puede calcular usando, por ejemplo, el algoritmo de reducción gaussiano.

$$s = x \cdot T'^{-1} \tag{1.12}$$

donde $x = (x^v, x^m)$ con x^v son los vinagres aleatorios y x^m los aceites que hemos calculado en la ecuación (1.8).

Algoritmo de verificación

Para comprobar que el mensaje es correcto y que no ha sufrido ninguna transformación durante el envío del mismo, se utiliza una versión del sistema utilizado para la firma. Se hace modificación para que el atacante no pueda obtener la clave privada ni las variables de aceite y vinagre. Para cada $k \in \{1, \ldots, m\}$ se tiene que cumplir la igualdad (1.13).

$$h_k = \alpha_{pub_k} s' + \beta_{pub_k} s' \tag{1.13}$$

Ejemplo

A continuación se muestra un ejemplo para mejor comprensión del algoritmo UOV utilizando valores de m y v pequeños, ambos con valor 3, y dejando fijo el valor de r igual a 7.

Generamos la clave privada con valores aleatorios. Notamos que las matrices del vector α_{priv}, ecuación 1.14, son matrices triangulares superiores de dimensión 3 x 6, esto es dimensión m x n. Además el número de matrices es 3 el valor de v.

$$\alpha_{priv} = \begin{bmatrix} \begin{bmatrix} 1\,0\,1\,1\,1\,0 \\ 0\,0\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1\,1 \end{bmatrix} \begin{bmatrix} 1\,1\,0\,1\,1\,0 \\ 0\,0\,1\,1\,1\,0 \\ 0\,0\,1\,1\,1\,1 \end{bmatrix} \begin{bmatrix} 0\,0\,0\,1\,0\,0 \\ 0\,0\,1\,1\,0\,1 \\ 0\,0\,0\,0\,1\,1 \end{bmatrix} \end{bmatrix} \tag{1.14}$$

La segunda parte de la clave privada es β_{priv}, consta de un vector de 3 vectores, donde cada uno tiene n componentes, esto es 6 componentes, ecuación 1.15.

$$\beta_{priv} = \begin{bmatrix} \begin{bmatrix} 1\,0\,0\,1\,0\,0 \end{bmatrix} \begin{bmatrix} 0\,0\,1\,0\,1\,0 \end{bmatrix} \begin{bmatrix} 1\,1\,0\,1\,0\,1 \end{bmatrix} \end{bmatrix} \tag{1.15}$$

La forma de la matriz T viene dada por la ecuación 1.9, la del ejemplo se muestra en la ecuación 1.16

$$T = \left[\begin{array}{c|c} I_3 & T_{3x3} \\ \hline 0 & I_3 \end{array}\right] = \left[\begin{array}{ccc|ccc} 1&0&0&0&0&0 \\ 0&1&0&0&0&1 \\ 0&0&1&1&0&1 \\ \hline 0&0&0&1&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{array}\right] \tag{1.16}$$

Clave pública consta de dos partes α_{pub} y β_{pub}. Cada componente de α_{pub} se calculan multiplicando tres matrices, una matriz, que tiene dos partes la I_3 y T'_{3x3}, por la $k-ésima$ componente de α_{priv} y por T.

$$\begin{aligned} \alpha_{pub_1} &= \left[\begin{array}{c} 1\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \\ \hline 0\,0\,1 \\ 0\,0\,0 \\ 0\,1\,1 \end{array}\right] \cdot \left[\begin{array}{c} 1\,0\,1\,1\,1\,0 \\ 0\,0\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1\,1 \end{array}\right] \cdot \left[\begin{array}{ccc|ccc} 1&0&0&0&0&0 \\ 0&1&0&0&0&1 \\ 0&0&1&1&0&1 \\ \hline 0&0&0&1&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{array}\right] \\ &= \left[\begin{array}{ccc|ccc} 1&0&1&1&1&0 \\ 0&0&0&0&1&1 \\ 0&0&0&1&1&1 \\ \hline 0&0&0&1&1&1 \\ 0&0&0&0&0&0 \\ 0&0&0&1&0&0 \end{array}\right] \cdot \left[\begin{array}{ccc|ccc} 1&0&0&0&0&0 \\ 0&1&0&0&0&1 \\ 0&0&1&1&0&1 \\ \hline 0&0&0&1&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{array}\right] \\ &= \begin{bmatrix} 1&0&1&0&1&1 \\ 0&0&0&0&1&1 \\ 0&0&0&1&1&1 \\ 0&0&0&1&1&1 \\ 0&0&0&0&0&0 \\ 0&0&0&1&0&0 \end{bmatrix} \end{aligned} \tag{1.17}$$

De la misma forma se obtiene α_{pub_2} y α_{pub_3}, así la ecuación 1.18 muestra el resultado de α_{pub}.

$$\alpha_{pub} = \left[\begin{bmatrix} 1&0&1&0&1&1 \\ 0&0&0&0&1&1 \\ 0&0&0&1&1&1 \\ 0&0&0&1&1&1 \\ 0&0&0&0&0&0 \\ 0&0&0&1&0&0 \end{bmatrix} \begin{bmatrix} 1&1&0&1&1&1 \\ 0&0&1&0&1&1 \\ 0&0&1&0&1&0 \\ 0&0&1&0&1&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&1 \end{bmatrix} \begin{bmatrix} 0&0&0&1&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&0&1&1 \\ 0&0&0&0&1&1 \\ 0&0&0&0&0&0 \\ 0&0&1&0&1&1 \end{bmatrix} \right] \tag{1.18}$$

Ahora se calcula β_{pub} para ello se va a realizar el cálculo de la primera componente del vector β_{pub_1} a modo de ejemplo, multiplicando la primera componente de β_{priv} por T, ecuación 1.19.

$$\beta_{pub_1} = \left[\begin{array}{c} 1\,0\,0\,1\,0\,0 \end{array}\right] \cdot \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \end{array}\right] \tag{1.19}$$

De manera análoga se calculan el resto de componentes, β_{pub_2} y β_{pub_3}, para obtener el vector completo β_{pub}, ecuación 1.20.

$$\beta_{pub} = \left[\left[1\,0\,0\,1\,0\,0 \right] \left[0\,0\,1\,1\,1\,1 \right] \left[1\,1\,0\,1\,0\,0 \right] \right] \tag{1.20}$$

Llegados a este punto comienza el proceso de firma del mensaje, en este caso el mensaje al que se le va a realizar la firma es "`Este mensaje es un mensaje de prueba. Quiero se sea un poco largo para que se aprecie el efecto de la función hash`", el hash de dicho mensaje calculado con la función `sha256` es "`c2f`", la ecuación 1.21 contiene el *hash* en GF(128).

$$hash = \left[\left[0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0 \right] \left[0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0 \right] \left[0,\ 0,\ 0,\ 1,\ 1,\ 1,\ 1 \right] \right] \tag{1.21}$$

Los vinagres se han tomado de forma aleatoria, ecuación 1.22.

$$a^{v} = \left[\left[0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 0 \right] \left[1,\ 1,\ 1,\ 1,\ 1,\ 0,\ 0 \right] \left[1,\ 1,\ 1,\ 1,\ 0,\ 0,\ 1 \right] \right] \tag{1.22}$$

La $k-ésima$ matriz del vector A es el resultado de multiplicar el vinagre por la matriz α_{priv_k} donde k se mueve entre 1 y 3, para lo cual, es necesario extender los componentes de las matrices de α_{priv}, que se encuentran en GF(2), al cuerpo GF(128). En el ejemplo no se ha puesto la matriz extendida para entender mejor la multiplicación y no perderse con los vectores, finalmente la multiplicación será el vector nulo si se multiplica por 0 o el mismo vector cuando se multiplique por 1.

$$\begin{aligned} A_1 &= \left[\left[0,1,1,0,1,0,0 \right] \left[1,1,1,1,1,0,0 \right] \left[1,1,1,1,0,0,1 \right] \right] \cdot \left[\begin{array}{c} 1\,0\,1\,1\,1\,0 \\ 0\,0\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1\,1 \end{array}\right] \\ &= \left[\left[0,1,1,0,1,0,0 \right] \left[0,0,0,0,0,0,0 \right] \left[0,1,1,0,1,0,0 \right] \right. \\ &\quad \left. \left[1,0,0,1,1,0,1 \right] \left[0,1,1,0,0,0,1 \right] \left[0,0,0,0,1,0,1 \right] \right] \end{aligned} \tag{1.23}$$

Calculando el resto de las componentes se obtiene la matriz A, ecuación 1.24.

$$\begin{aligned} A = [\ [\ [\ 0,1,1,0,1,0,0\ ][\ 0,0,0,0,0,0,0\ ][\ 0,1,1,0,1,0,0\ ] \\ [\ 1,0,0,1,1,0,1\ ][\ 0,1,1,0,0,0,1\ ][\ 0,0,0,0,1,0,1\ ]\ ] \\ [\ [\ 0,1,1,0,1,0,0\ ][\ 0,1,1,0,1,0,0\ ][\ 0,0,0,0,1,0,1\ ] \\ [\ 0,1,1,0,0,0,1\ ][\ 0,1,1,0,0,0,1\ ][\ 1,1,1,1,0,0,1\ ]\ ] \\ [\ [\ 0,0,0,0,0,0,0\ ][\ 0,0,0,0,0,0,0\ ][\ 1,1,1,1,1,0,0\ ] \\ [\ 1,0,0,1,0,0,0\ ][\ 1,1,1,1,0,0,1\ ][\ 0,0,0,0,1,0,1\ ]\ ]\ ] \end{aligned} \tag{1.24}$$

Los coeficientes se obtienen de la suma de las últimas m componenetes, en este caso las 3 últimas, de cada A_k con β_{priv_k}. Pero tenemos el mismo problema de antes, puesto que los coeficientes de A_k pertenecen a GF(128) y los coeficientes de β_{priv_k} se encuentra en GF(2), por tanto lo que vamos a sumar es 0 o 1, o lo que es lo mismo realizar la operación lógica XOR con los elementos $[0,0,0,0,0,0,0]$ o $[0,0,0,0,0,0,1]$, ecuación 1.25.

$$coef = \left[\left[\begin{bmatrix} 1, 0, 0, 1, 1, 0, 0 \\ 0, 1, 1, 0, 0, 0, 1 \\ 0, 0, 0, 0, 1, 0, 1 \end{bmatrix} \right] \left[\begin{bmatrix} 0, 1, 1, 0, 0, 0, 1 \\ 0, 1, 1, 0, 0, 0, 0 \\ 1, 1, 1, 1, 0, 0, 1 \end{bmatrix} \right] \left[\begin{bmatrix} 1, 0, 0, 1, 0, 0, 1 \\ 1, 1, 1, 1, 0, 0, 1 \\ 0, 0, 0, 0, 1, 0, 0 \end{bmatrix} \right] \right] \tag{1.25}$$

Los términos corresponden a la parte de la derecha de la ecuación 1.7, a continuación se va a realizar el cálculo para k igual 1. La ecuación 1.26 corresponde al primer vector de β, $[\ 1\ 0\ 0\ 1\ 0\ 0\ ]$, donde se han transformado los elementos de GF(2) a GF(128) y se han tomado las 3 primeras componentes.

$$\beta_1^v = [\ [\ 0,0,0,0,0,0,1\ ][\ 0,0,0,0,0,0,0\ ][\ 0,0,0,0,0,0,0\ ]\ ] \tag{1.26}$$

De la misma forma se han tomado las 3 primeras componenetes del primer vector de A, ecuación 1.27.

$$A_1^v = [\ [\ 0,1,1,0,1,0,0\ ][\ 0,0,0,0,0,0,0\ ][\ 0,1,1,0,1,0,0\ ]\ ] \tag{1.27}$$

El vector sum contiene la suma de A_1^v y β_1^v, ecuación 1.28.

$$sum = [\ [\ 0,1,1,0,1,0,1\ ][\ 0,0,0,0,0,0,0\ ][\ 0,1,1,0,1,0,0\ ]\ ] \tag{1.28}$$

Finalmente, para calcular $term$ hay que multiplicar sum por a^v y restarselo a la primera componente de $hash$. Al realizar las operaciones módulo 2, la resta y la suma son lo mismo, por tanto se realizará la operación lógica *XOR* con el $hash$, ecuación1.29.

$$\begin{aligned}
term_1 &= [[0,1,1,0,1,0,1]\,[0,0,0,0,0,0,0]\,[0,1,1,0,1,0,0]] \cdot \begin{bmatrix} [0,1,1,0,1,0,0] \\ [1,1,1,1,1,0,0] \\ [1,1,1,1,0,0,1] \end{bmatrix} + \begin{bmatrix} 1,0,1,1,0,0,0 \end{bmatrix} \\
&= \begin{bmatrix} 0,0,0,1,1,0,0 \end{bmatrix} + \begin{bmatrix} 1,0,1,1,0,0,0 \end{bmatrix} \\
&= \begin{bmatrix} 1,0,1,0,1,0,0 \end{bmatrix}
\end{aligned} \tag{1.29}$$

De la misma forma se calculan el resto de los términos llegando al vector $term$, ecuación 1.30.

$$term = \begin{bmatrix} \begin{bmatrix} 1,0,1,0,1,0,0 \end{bmatrix} \begin{bmatrix} 1,1,0,1,0,0,0 \end{bmatrix} \begin{bmatrix} 1,1,0,0,0,0,0 \end{bmatrix} \end{bmatrix} \tag{1.30}$$

Tras la resolución del sistema $coef \cdot oil = term$ se han obtenido los aceites, ecuación 1.31.

$$oil = \begin{bmatrix} \begin{bmatrix} 0,0,0,0,1,1,0 \end{bmatrix} \begin{bmatrix} 0,0,1,1,0,1,1 \end{bmatrix} \begin{bmatrix} 0,1,0,0,1,1,0 \end{bmatrix} \end{bmatrix} \tag{1.31}$$

Ahora se forma el vector aux uniendo los vinagres con los aceites, ecuación 1.32.

$$\begin{aligned}
aux = [\; & \begin{bmatrix} 0,0,0,0,1,1,0 \end{bmatrix} \begin{bmatrix} 0,0,1,1,0,1,1 \end{bmatrix} \begin{bmatrix} 0,1,0,0,1,1,0 \end{bmatrix} \\
& \begin{bmatrix} 0,1,1,0,1,0,0 \end{bmatrix} \begin{bmatrix} 1,1,1,1,1,0,0 \end{bmatrix} \begin{bmatrix} 1,1,1,1,0,0,1 \end{bmatrix} \;]
\end{aligned} \tag{1.32}$$

La firma del mensaje se obtiene multiplicando el vector aux por la inversa de T, que de la forma que está construida dicha matriz T coincide con su inversa, ecuación 1.33.

$$\begin{aligned}
firma = [\; & \begin{bmatrix} 0,1,1,0,1,0,0 \end{bmatrix} \begin{bmatrix} 1,0,1,1,0,1,0 \end{bmatrix} \begin{bmatrix} 1,0,1,1,0,0,1 \end{bmatrix} \\
& \begin{bmatrix} 0,0,0,0,1,1,0 \end{bmatrix} \begin{bmatrix} 0,0,1,1,0,1,1 \end{bmatrix} \begin{bmatrix} 0,1,0,0,1,1,0 \end{bmatrix} \;]
\end{aligned} \tag{1.33}$$

Para realizar la confirmación del mensaje es necesario calcular dos variables auxiliares, α_{aux} y β_{aux}.

Para calcular α_{aux1} hay que multiplicar los vectores $firma$, α_{pub_1}y $firma$

transpuesta, ecuación 1.34.

$$\begin{aligned} \alpha_{aux1} &= \left[\ firma\ \right] \cdot \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \cdot \left[\ trans(firma)\ \right] \\ &= \left[\ 0,1,1,1,1,1,0\ \right] \end{aligned} \tag{1.34}$$

La segunda variable $\beta_{aux k}$ es el producto de la $k-ésima$ componente del vector β_{pub}, donde sus componenetes se encuentran en GF(2) y habrá que transformarlos en componentes de GF(128), por el vector $firma$ transpuesta, ecuación 1.35.

$$\begin{aligned} \beta_{aux1} &= \left[\ 1\,0\,0\,1\,0\,0\ \right] \cdot \left[\ trans(firma)\ \right] \\ &= \left[\ 0,1,1,1,1,1,0\ \right] \end{aligned} \tag{1.35}$$

Calculando todas la componentes de α_{aux} y β_{aux} se obtienen los vectores las ecuaciones 1.36 y 1.37, respectivamente.

$$\alpha_{aux} = \left[\ \left[\ 0,1,1,1,1,1,0\ \right] \left[\ 1,1,0,0,0,0,0\ \right] \left[\ 1,1,0,0,1,1,1\ \right]\ \right] \tag{1.36}$$

$$\beta_{aux} = \left[\ \left[\ 0,1,1,0,0,1,0\ \right] \left[\ 1,1,0,0,0,1,0\ \right] \left[\ 1,1,0,1,0,0,0\ \right]\ \right] \tag{1.37}$$

Sumando los vectores componente a componente, ecuación 1.38, anteriormente calculados, se obtiene vector a comparar con el $hash$ del mensaje.

$$\left[\ \left[\ 0,0,0,1,1,0,0\ \right] \left[\ 0,0,0,0,0,1,0\ \right] \left[\ 0,0,0,1,1,1,1\ \right]\ \right] \tag{1.38}$$

Si el mensaje no ha sido corrompido habrá de ser igual que el $hash$ del mensaje, ecuación 1.39.

$$\begin{aligned} &\left[\ \left[\ 0,0,0,1,1,0,0\ \right] \left[\ 0,0,0,0,0,1,0\ \right] \left[\ 0,0,0,1,1,1,1\ \right]\ \right] == \\ &\left[\ \left[\ 0,0,0,1,1,0,0\ \right] \left[\ 0,0,0,0,0,1,0\ \right] \left[\ 0,0,0,1,1,1,1\ \right]\ \right] \end{aligned} \tag{1.39}$$

Capítulo 2

Planificación y costes

En este capítulo se van a definir las diferentes etapas del proyecto mediante diagramas de Gantt realizados con la aplicación *OpenProj*. Además se incluye el presupuesto necesario para la realización de dicho proyecto.

2.1. Planificación

La figura 2.1 muestra la propuesta inicial de las etapas del proyecto, donde se diferencian varios bloques, el primero sería el de estudio e introducción a lo que se va a realizar en el proyecto, que comprendería desde septiembre hasta finales de noviembre, el segundo bloque o implementación del algoritmo, desde mediados de noviembre hasta finales de diciembre, el tercer bloque contiene todo lo referente a la *blockchain* ARK siendo el grueso del proyecto, comienza a finales de enero hasta mayo. Y por último la parte de las de las pruebas, son 10 días en mayo. La memoria se ha redactado durante todo el proyecto.

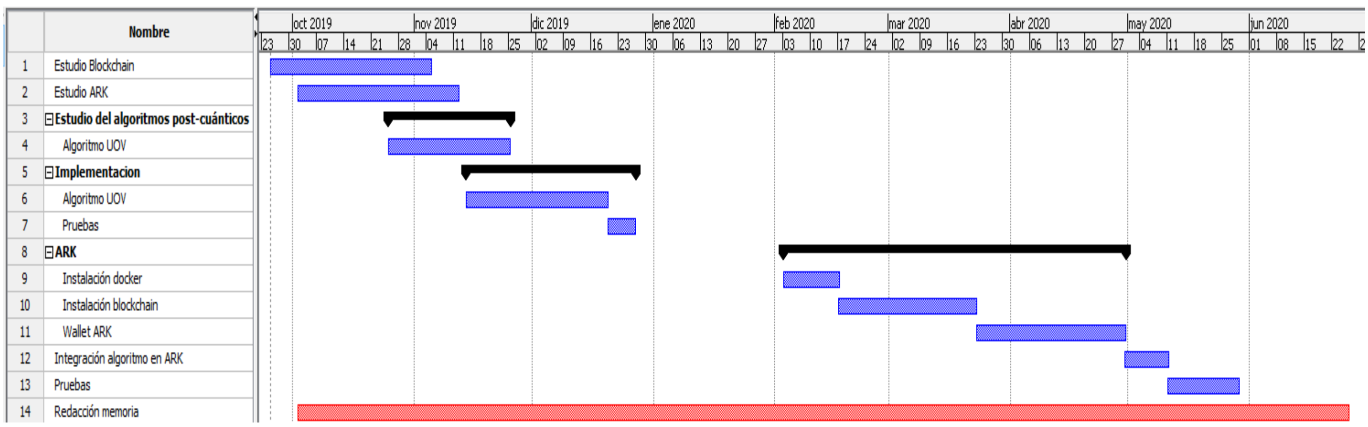

Figura 2.1: Digrama de Gantt inicial

Pero no todo ha sido como se había planificado, puesto que han surgido algunos imprevistos. A la hora de realizar la implementación del algoritmo UOV en `python` no existe una biblioteca para trabajar con matrices y cuerpos finitos al mismo tiempo, así ha aumentado el tiempo que se iba a dedicar al algoritmo. Además el trabajo con ARK ha sido más tedioso del esperado, retrasando los tiempos programados.

El diagrama de Gantt real se ha dividido en aproximadamente cuatrimestres para que se visualice mejor, la imagen 2.2 muestra las fases de estudio e implementación hasta febrero, la imagen 2.3 incluye el tiempo dedicado a terminar la implementación y realizar el trabajo con ARK hasta julio y la imagen 2.4 contiene el tiempo restante de trabajo con ARK desde julio hasta noviembre, además del periodo de pruebas.

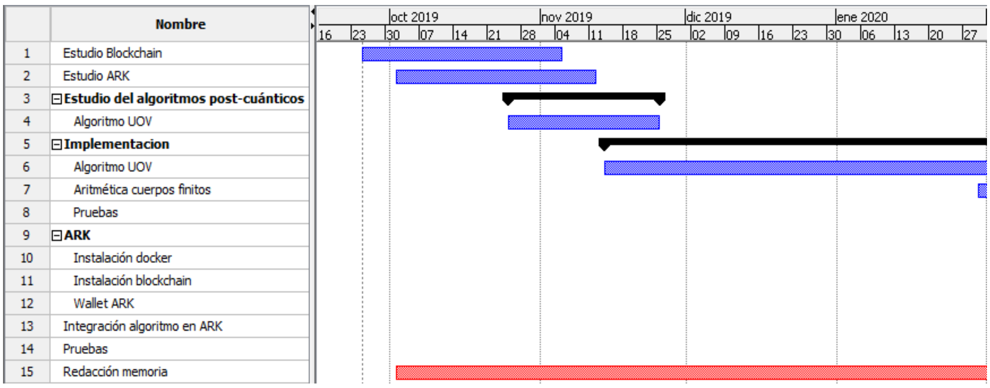

Figura 2.2: Diagrama de Gantt real. Parte I

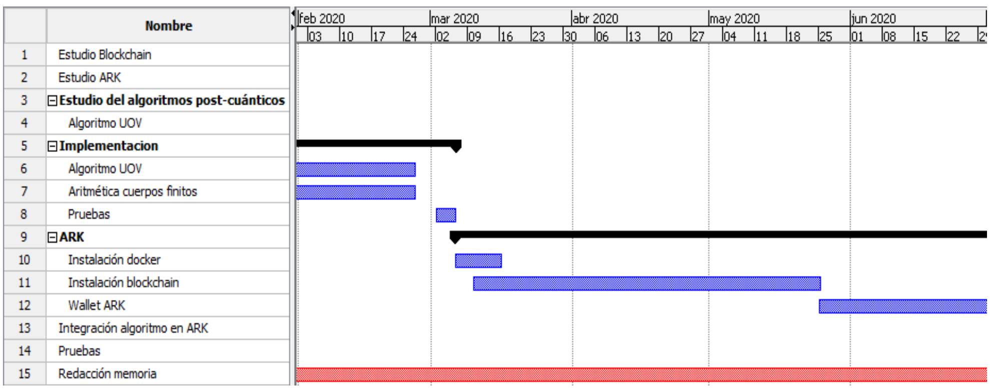

Figura 2.3: Diagrama de Gantt real. Parte II

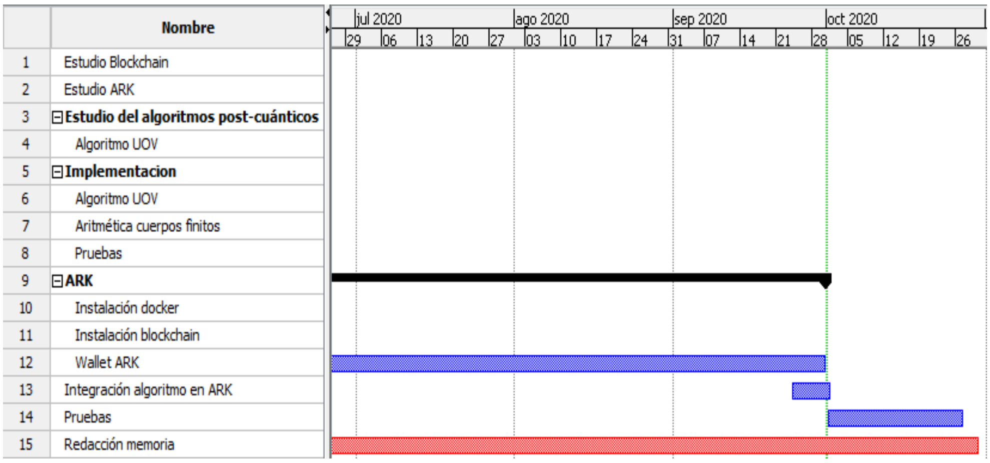

Figura 2.4: Diagrama de Gantt real. Parte III

2.2. Costes

A continuación se va a detallar el presupuesto invertido en el desarrollo del proyecto. Los costes del mismo se van a desglosar en costes indirectos, recursos humanos, inventariales y viajes.

Los costes indirectos corresponden a la electricidad, material de oficina y servicio de mantenimiento. Este último incluye el precio de un nuevo disco de memoria interno, ya que ha sido necesario aumentar la memoria del portátil durante el desarrollo del proyecto, veáse la tabla 2.1.

Descripción del coste	**Cantidad**
Electricidad	100€
Material de oficina	30€
Servicio de mantenimiento	45€
TOTAL (€)	175 €

Tabla 2.1: Desglose de los costes indirectos

La tabla 2.2 muestra los costes de recursos humanos dividos en tres grandes bloques, estudio, desarrollo y uno general, en el que corresponde al tiempo dedicado a la memoria y a las reuniones con ambos tutores. Se ha estimado que el precio por cada hora de trabajo es de 12€.

Descripción del coste	Cantidad (horas)	Coste total (€)
Bloque de estudio		
Estudio tecnología blockchain	32	384
Estudio ARK	50	600
Estudio algoritmo post-cuánticos	28	336
Bloque de desarrollo		
Algoritmo UOV	15	180
Aritmetica cuerpos finitos	23	276
Instalación docker	20	240
Instalación *blockchain*	20	240
Wallet ARK	25	300
Integración algoritmo	10	120
Pruebas	16	192
Bloque general		
Redacción memoria	67	804
Corrección memoria	19	228
Reuniones	35	420
TOTAL (€)	4.320	

Tabla 2.2: Desglose de los costes en recursos humanos

El equipo de trabajo ha sido con un ASUS TP300L, teniendo en cuenta que el precio de adquisición del ordenador fue de 750,00€ y suponiendo que la vida media de un ordenador es de unos 6 años, entonces el gasto que corresponde durante los 13 meses del desarrollo del proyecto es de 135,41€.

Por último tenemos los costes en viajes a la escuela de informática para las reuniones con el tutor, puesto que a la facultad de ciencias no era necesario coger un transporte. Solo se han realizado viajes en el primer cuatrimestre del curso 2019-2020, ya que el resto de las reuniones han sido de forma virtual, obteniendo un total de costes de 22€.

Tipo de costes	**Cantidad**
Indirectos	175€
Recursos humanos	4.320€
Inventariales	135,41€
Viajes	22€
TOTAL (€)	4.652,41€

Tabla 2.3: Presupuesto total desglosado

Una vez que hemos calculado todos los gastos posibles, tabla 2.3, vamos a añadir un margen de contingencia del 5 % para posibles imprevistos del proyecto. Hemos obtenido que los gastos previstos serán de 4.652,41€, por tanto el margen de contigencia será de 232,62€, así el presupuesto final sera de 4.885,03€.

Capítulo 3

Análisis del problema

3.1. Especificación de requisitos

Debe incluir una clara descripción de las funcionalidades que se esperan alcanzar, así como las restricciones o condicionantes que puedan determinar el diseño o solución adoptada. Tras eso deben especificarse claramente los requisitos.

Los requisitos pueden ser funcionales (e.g. La herramienta debe mostrar las medidas de la red en tiempo real), o no funcionales (e.g. se debe garantizar el acceso seguro a la herramienta; el rendimiento debe ser alto; el consumo de memoria debe ser bajo; etc.)

3.2. Análisis

El objetivo de este apartado es mostrar en diferentes subapartados los diferentes subproblemas que han aparecido al realizar el proyecto, describiendo las alternativas que se han considerado, y justificando las decisiones que se han adoptado. A veces, especialmente cuando los conceptos utilizados en este apartado son extensos, es necesario clarificarlos previamente en el Capítulo de *Introducción* (sección *Contenidos teóricos para la comprensión del proyecto*).

Capítulo 4

Diseño

4.1. Algoritmo criptográfico

4.2. Ecosistema ARK

Capítulo 5

Implementación

Aquí se proporcionan los detalles de las funciones implementadas para la ejecución algoritmo UOV, además de la implementación de la aritmética en el cuerpo de 128 elementos.

5.1. Funciones del cuerpo de 128 elementos

Las funciones referentes a esta sección son la suma, el producto y conversiones de elementos del cuerpo de 2^7 elementos a elementos del cuerpo de 128 elementos.

Los elementos del cuerpo se representarán con vectores de siete componentes para poder facilitar la implementación de la suma y del producto. Esto se refleja en las dos tablas denominadas `exp` y `log`, que tienen la estructura que se muestra en los códigos 5.1 y 5.2, respectivamente. Para la implementación se han usado las variables diccionario de `python`, puesto que tienen fácil acceso a todas las componentes.

```
exp = {
   0 : [0, 0, 0, 0, 0, 0, 1],
   1 : [0, 0, 0, 0, 0, 1, 0],
   125 : [1, 1, 0, 0, 0, 0, 1],
   126 : [1, 0, 0, 0, 0, 0, 1]
}
```

Código 5.1: Tabla para calcular la potencia en el cuerpo

```
log = {
   tuple( [0, 0, 0, 0, 0, 0, 1] ) : ' 0 ',
   tuple( [0, 0, 0, 0, 0, 1, 0] ) : ' 1 ',
   tuple( [1, 1, 0, 0, 0, 0, 1] ) : ' 125 ',
   tuple( [1, 0, 0, 0, 0, 0, 1] ) : ' 126 ',
}
```

Código 5.2: Tabla para calcular el logaritmo en el cuerpo

La suma de dos elementos del cuerpo se ha implementando la puerta xor, esto es si las $iésimas$-componentes son iguales entonces la suma vale 0, en caso contrario vale 1, código 5.3.

Tipo	Nombre	Variable	Descripción
int vector	n1	Input	Elemento para sumar del cuerpo de 128 elementos
int vector	n2	Input	Elemento para sumar del cuerpo de 128 elementos
int vector	suma	Output	Elemento del cuerpo que almacena la suma de n1 y n2

Tabla 5.1: Parámetros de la función `suma`

```
def suma(n1=[], n2=[]):

   suma = []
   for i in range(len(n1)):
      if n1[i] == n2[i]:
         suma += [0]
      else:
         suma += [1]
   return suma
```

Código 5.3: Suma de dos elementos del cuerpo

Para el producto de dos elementos del cuerpo se ha diferenciado el caso en el que uno de los vectores sea 0 en dicho caso el producto vale 0. Si ninguno de los vectores es 0 entonces se hace uso de las tablas 5.1 y 5.2, para trabajar con enteros módulo 127, código 5.4.

Tipo	Nombre	Variable	Descripción
int vector	n1	Input	Elemento para multiplicar del cuerpo de 128 elementos
int vector	n2	Input	Elemento para multiplicar del cuerpo de 128 elementos
int	suma	Inner	Almacena la suma de los logaritmos de n1 y n2 módulo 127
int vector	product	Output	Elemento del cuerpo que almacena el producto de n1 y n2

Tabla 5.2: Parámetros de la función `product`

```
def product(n1=[], n2=[]):

   product = []
   if (n1 == [0,0,0,0,0,0,0]) or (n2 == [0,0,0,0,0,0,0]):
      product = [0,0,0,0,0,0,0]
   else:
```

```
        suma = (int(log.get(tuple(n1))) + int(log.get(tuple(n2))))%127
        product = exp.get(suma)
    return product
```

Código 5.4: Producto de dos elementos del cuerpo

Las siguientes funciones son necesarias para la resolución del sistema de ecuaciones.

Calcula el vector correspondiente vector en el cuerpo de un entero, código 5.5.

Tipo	Nombre	Variable	Descripción
int	n1	Input	Entero del que se desea calcular su vector en el cuerpo
int vector		Output	Elemento del cuerpo de 128 elementos

Tabla 5.3: Parámetros de la función `F128`

```
def F128(n1):

    if n1 == 127:
        return [0,0,0,0,0,0,0]
    else:
        return exp.get(n1)
```

Código 5.5: Convierte un entero en un elemento del cuerpo

Al estar en un cuerpo todo elementos tiene inverso y tiene sentido hacer la función `inverso` de un elemento del cuerpo, la implementación que se ha hecho ha sido hacer la multiplicación por los restantes elementos y comprobando que de como resultado el elemento unidad, en este caso $[0,0,0,0,0,0,1]$, código 5.6.

Tipo	Nombre	Variable	Descripción
int vector	n	Input	Elemento para calcular su inverso en el cuerpo de 128 elementos
int vector		Output	Inverso de n

Tabla 5.4: Parámetros de la función `inverse`

```
def inverse(n=[]):

    i = 0
    while product(n, F128(i)) != [0,0,0,0,0,0,1]:
        i += 1
    return F128(i)
```

Código 5.6: Inverso de un elemento del cuerpo

La función `mayor` compara si el elemento $n1$ es mayor que el elemento $n2$, para ello se convierten los vectores a enteros con la tabla `log` y se compara que entero es mayor, código 5.7.

Tipo	Nombre	Variable	Descripción
int vector	n1	Input	Elemento a comparar con n2 en el cuerpo de 128 elementos
int vector	n2	Input	Elemento a comparar con n1 en el cuerpo de 128 elementos
int	log1	Inner	Almacena el logaritmo de n1
int	log2	Inner	Almacena el logaritmo de n2
boolean		Output	`True` si n1 es mayor `False` en otro caso

Tabla 5.5: Parámetros de la función `mayor`

```
def mayor (n1=[],n2=[]):

   if n1 == [0,0,0,0,0,0,0]:
      return False
   else:
      log1, log2 = log.get(tuple(n1)), log.get(tuple(n2))
      return int(log1) > int(log2)
```

Código 5.7: Compara dos elementos del cuerpo

Convierte una matriz $n1$ del cuerpo $\mathbb{F}_2$ en un vector n del cuerpo de 128 elementos. Como los elementos de la matriz son solo 0 y 1 la conversión se reduce a, si en la matriz hay un 0 añade al vector n el vector $[0,0,0,0,0,0,0]$ y en otro caso $[0,0,0,0,0,0,1]$, código 5.8.

Tipo	Nombre	Variable	Descripción
int vector	n1	Input	Matriz con elementos en $\mathbb{F}_2$
int	row	Inner	Almacena las filas generadas de la matriz n
int vector	n	Output	Matriz en el cuerpo de 128 elementos

Tabla 5.6: Parámetros de la función `matrix_F2to128`

```
def matrix_F2to128(n1=[]):

   n =[]
   for i in range(len(n1)):
      row =[]
      for j in range(len(n1[0])):
         if n1[i][j] == 0:
```

```
            row += [[0,0,0,0,0,0,0]]
        else:
            row += [[0,0,0,0,0,0,1]]
    n += [row]
return n
```

Código 5.8: Matriz de $\mathbb{F}_2$ a un elemento del cuerpo 128 elementos

Convierte un vector de matrices $n1$ del cuerpo $\mathbb{F}_2$ en una matriz $matrix$ del cuerpo de 128 elementos. Como en la función anterior los elementos de la matriz son solo 0 y 1, se aplica el mismo cambio, código 5.9.

Tipo	**Nombre**	**Variable**	**Descripción**
int vector	n1	Input	Vector de matriz con elementos en $\mathbb{F}_2$
int	n	Inner	Almacena las matrices del vector matrix
int	row	Inner	Almacena las filas generadas de la matriz n
int vector	matrix	Output	Vector de matrices con elementos en el cuerpo de 128 elementos

Tabla 5.7: Parámetros de la función `matrix3d_F2to128`

```
def matrix3d_F2to128(n1=[]):

    matrix =[]
    for i in range(len(n1)):
        n = []
        for j in range(len(n1[0])):
            row = []
            for k in range(len(n1[0][0])):
                if n1[i][j][k] == 0:
                    row += [[0,0,0,0,0,0,0]]

                else:
                    row += [[0,0,0,0,0,0,1]]
            n += [row]
        matrix += [n]
    return matrix
```

Código 5.9: Vector de matrices de $\mathbb{F}_2$ a una matriz del cuerpo de 128 elementos

5.2. Funciones con matrices

En esta sección de explicarán funciones como la suma de matrices, cálculo de la matriz transpuesta, matriz identidad y el producto de matrices tanto en $\mathbb{F}_2$ como en el cuerpo de 128 elementos. Además incluye la función que resuelve el sistema de ecuaciones con el método de Gauss-Jordan.

Suma de dos matrices $m1$ y $m2$ del cuerpo de 128 elementos, código 5.10.

Tipo	Nombre	Variable	Descripción
int vector	m1	Input	Matriz a sumar en el cuerpo de 128 elementos
int vector	m2	Input	Matriz a sumar en el cuerpo de 128 elementos
int	row	Inner	Almacena las filas de n
int vector	m_suma	Output	Suma de las matrices m1 y m2 en el cuerpo de 128 elementos

Tabla 5.8: Parámetros de la función `matrix_sum`

```
def matrix_sum (m1, m2):

   m_suma = []

   if (len(m1) == len(m2)) and (len(m1[0]) == len(m2[0])):
      for i in range(len(m1)):
         row = []
         for j in range(len(m1[0])):
            row += [suma (m1[i][j], m2[i][j])]
         m_suma += [row]
   return m_suma
```

Código 5.10: Suma de dos matrices con elementos en el cuerpo

Producto de dos matrices con elementos en el cuerpo de 128 elementos, código 5.11.

Tipo	Nombre	Variable	Descripción
int vector	n1	Input	Matriz a multiplicar en el cuerpo de 128 elementos
int vector	n2	Input	Matriz a multiplicar en el cuerpo de 128 elementos
int	p_suma	Inner	Almacena la suma de los productos de cada elemento de la fila de n1 y de la columna de n2
int	row	Inner	Almacena las filas de prod
int vector	prod	Output	Producto de las matrices m1 y m2 en el cuerpo de 128 elementos

Tabla 5.9: Parámetros de la función `matrix_product`

```
def matrix_product (n1=[], n2=[]):

   prod=[]
   if len(n1[0]) == len(n2):
      for i1 in range(len(n1)):
         row = []
         for j in range(len(n2[0])):
            p_suma = [0,0,0,0,0,0,0]
            for i2 in range(len(n2)):
               p_suma = suma (product(n1[i1][i2], n2[i2][j]), p_suma)
            row += [p_suma]
```

```
        prod += [row]
    return prod
```

Código 5.11: Producto de matrices con elementos del cuerpo

Producto de dos matrices con elementos en el cuerpo $\mathbb{F}_2$, como los elementos de la matriz son 0 y 1 la suma acomulada se hace con la operación lógica `xor`, código 5.12.

Tipo	Nombre	Variable	Descripción
int vector	n1	Input	Matriz a multiplicar en $\mathbb{F}_2$
int vector	n2	Input	Matriz a multiplicar en $\mathbb{F}_2$
int	p_suma	Inner	Almacena la suma de los productos de cada elemento de la fila de n1 y de la columna de n2
int	row	Inner	Almacena las filas de prod
int vector	prod	Output	Producto de las matrices m1 y m2 en $\mathbb{F}_2$

Tabla 5.10: Parámetros de la función `matrix_product_F2`

```
def matrix_product_F2 (n1=[], n2=[]):

    prod=[]
    if len(n1[0]) == len(n2):
        for i1 in range(len(n1)):
            row = []
            for j in range(len(n2[0])):
                p_suma = 0
                for i2 in range(len(n2)):
                    p_suma = (n1[i1][i2] * n2[i2][j]) ^ p_suma
                row += [p_suma]
            prod += [row]
    return prod
```

Código 5.12: Producto de matrices con elementos en $\mathbb{F}_2$

Calcula la matriz transpuesta de la matriz dada como parámetro, código 5.13.

Tipo	Nombre	Variable	Descripción
int vector	m	Input	Matriz para calcular su transpuesta
int	row	Inner	Almacena las filas de trans
int vector	trans	Output	Matriz transpuesta

Tabla 5.11: Parámetros de la función `matrix_transpose`

```
def matrix_transpose(m):
```

```

    trans = []
    for j in range(len(m[0])):
        row = []
        for i in range(len(m)):
            row += [m[i][j]]
        trans += [row]
    return trans
```

Código 5.13: Matriz transpuesta

Calcula la matriz identidad con una determinada dimensión, dim, que se pasa como parámetro, código 5.14.

Tipo	Nombre	Variable	Descripción
int	dim	Input	Dimensión de la que calcular la matriz identidad en el cuerpo de 128 elementos
int	row	Inner	Almacena las filas de matrix
int vector	matrix	Output	Matriz identidad

Tabla 5.12: Parámetros de la función `matrix_identity`

```
def matrix_identity(dim):

    matrix = []

    for i in range(dim):
        row = []
        for j in range(dim):
            if i == j:
                row += [[0,0,0,0,0,0,1]]
            else:
                row += [[0,0,0,0,0,0,0]]
        matrix += [row]
    return matrix
```

Código 5.14: Matriz identidad del cuerpo

La función `matrix_rref` resuelve de un sistema de ecuaciones con el método de Gauss-Jordan, código 5.15. El sistema de ecuaciones es de la forma de la ecuación 5.1.

$$A \cdot x = b \tag{5.1}$$

```
def matrix_rref(A, b):

    r = 0
    row = []
    n = len(A)

    #Matriz aumentada, añadir b como una columna
    M = A
```

Tipo	**Nombre**	**Variable**	**Descripción**
int vector	A	Input	Matriz de la ecuación
int vector	b	Input	Vector de la ecuación
int vector	M	Inner	Matriz aumentada, combinación de A y b
int vector	x	Output	Solución del sistema

Tabla 5.13: Parámetros de la función `matrix_rref`

```
    for i in range(len(M)):
        M[i] += b[i]

    for k in range(n):
        #Intercambio de filas para que quede arriba la de menor valor
        for i in range(k, n):
            if mayor(M[i][k], M[k][k]):
                row = M[k]
                M[k] = M[i]
                M[i] = row

        #Hacer ceros
        for j in range(k+1, n):
            q = product(M[j][k], inverse(M[k][k]))
            for m in range(k, n+1):
                M[j][m] = suma(M[j][m], product(q, M[k][m]))

    #Calcular la solución x de abajo arriba
    x = [[[0,0,0,0,0,0,0]] for i in range(n)]

    x[n-1] = [product(M[n-1][n], inverse(M[n-1][n-1]))]
    for i in range(n-1, -1, -1):
        z = [0,0,0,0,0,0,0]
        for j in range(i+1, n):
            z = suma(z, product(M[i][j], x[j][0]))
        x[i] = [product(suma(M[i][n], z), inverse(M[i][i]))]

    return x
```

Código 5.15: Método de Gauss-Jordan

5.3. Funciones algoritmo UOV

La función `clavePrivada` genera la clave privada de un usuario, para ello genera una matriz triangular superior, α, en $\mathbb{F}_2$ con valores aleatorios, y un vector, β, en $\mathbb{F}_2$. Los parámetros m y v son el número de variables de aceite y vinagre, respectivamente, código 5.16.

```
def clavePrivada (m, v):

    alpha, beta = [], []
    n = m+v

    for k in range(m):
        #alpha matriz triangular superior
```

Tipo	Nombre	Variable	Descripción
int vector	m	Input	Número de aceites
int vector	v	Input	Número de vinagres
int vector	alpha	Output	Vector de matrices triangulares superiores aleatorias en $\mathbb{F}_2$, parte de la clave privada
int vector	beta	Output	Matriz aleatoria en $\mathbb{F}_2$, parte de la clave privada

Tabla 5.14: Parámetros de la función `clavePrivada`

```
    aux = []
    for i in range (v):
       row = []
       for j in range (n):
          if i > j:
             row += [0]
          else:
             row += [randint(0,1)]
       aux += [row]
    alpha += [aux]

    #beta
    row = []
    for i in range(n):
       if randint(0,1) == 0:
          row += [randint(0,1)]

    beta += [row]
 return alpha, beta
```

Código 5.16: Generación clave privada

La función `clavePublica` genera la clave pública a partir de la clave privada, calculada con la función anterior, los valores m y v el número de variables de aceite y vinagre, además de una matriz de distorsión, código 5.17.

Tipo	Nombre	Variable	Descripción
int vector	m	Input	Número aceites
int vector	v	Input	Número vinagres
int vector	alpha	Input	Vector de matrices triangulares superiores en $\mathbb{F}_2$, parte de la clave privada
int vector	beta	Input	Matriz en $\mathbb{F}_2$, parte de la clave privada
int vector	T	Input	Matriz de distorsión de la forma 1.9
int vector	alpha_pub	Output	Vector de matrices en $\mathbb{F}_2$, parte de la clave pública
int vector	beta_pub	Output	Matriz en $\mathbb{F}_2$, parte de la clave pública

Tabla 5.15: Parámetros de la función `clavePublica`

```
def clavePublica(m, v, alpha, beta, T):

   alpha_pub = []
   beta_pub = []
   T_trans = matrix_transpose(T[0:v])

   for k in range(len(alpha)):
      aux = matrix_product_F2(T_trans , alpha [k])
      alpha_pub += [matrix_product_F2(aux, T)]


      beta_pub += [matrix_product_F2([beta[k]], T)]

   return alpha_pub, beta_pub
```

Código 5.17: Generación clave pública

La función `signature` calcula la firma del hash de un mensaje, $hashed$, con las claves privadas del usuario, $alpha\_F2$ y $beta\_F2$, el número de aceites y vinagres, m y v, el tamaño del cuerpo, r y con la matriz de transición, T. Se toman los vinagres aleatorios, se resuelve el sistema para calcular los aceites, y con la ecuación 1.12 obtenemos la firma del mensaje, código 5.18.

Tipo	**Nombre**	**Variable**	**Descripción**
int vector	hashed	Input	Vector *hash*
int vector	alpha_F2	Input	Vector de matrices triangulares superiores en $\mathbb{F}_2$, parte de la clave privada
int vector	beta_F2	Input	Matriz en $\mathbb{F}_2$, parte de la clave privada
int	m	Input	Número aceites
int	r	Input	Dimensión de
int	v	Input	Número vinagres
int vector	T	Input	Matriz de distorsión de la forma 1.9
int vector	alpha	Inner	Almecena los correspondientes valores de alpha_F2 en el cuerpo de 128 elementos
int vector	beta	Inner	Almecena los correspondientes valores de beta_F2 en el cuerpo de 128 elementos
int vector	vinagre	Inner	Vector aleatorio de tamaño v, contiene los vinagres
int vector	coef	Inner	Almacena los coeficientes del sistema
int vector	term	Inner	Almacena los términos del sistema
int vector	oil	Inner	Almacena la solución del sistema, son los aceites del algoritmo
int vector	firma	Output	Contiene la firma de *hashed*

Tabla 5.16: Parámetros de la función `signature`

```
def signature(hashed, alpha_F2, beta_F2, m, r, v, T):

   alpha = matrix3d_F2to128(alpha_F2)
```

```
    beta = matrix_F2to128(beta_F2)

    vinagre = []
    for k in range(v):
       aux = randint(0, 127)
       vinagre += [F128(aux)]
    coef = []
    term = []

    n= m + v
    for k in range(m):
       A = matrix_product([vinagre], alpha[k])
       coef += matrix_sum([A[0][v:n]], [beta[k][v:n]])
       v_suma = suma (matrix_product([A[0][0:v]], matrix_transpose([vinagre
  ]))[0][0], matrix_product([beta[k][0:v]], matrix_transpose([vinagre]))
  [0][0])
       term += [suma(hashed[k], v_suma)]


    oil = matrix_rref(coef, matrix_transpose([term]))

    aux = []
    aux += vinagre + matrix_transpose(oil)[0]
    firma = matrix_product([aux], matrix_transpose(matrix_F2to128(T))) #T =
      T.inverse()
    return firma[0]
```

Código 5.18: Firma del mensaje

Por último la función `verify` comprueba si la firma, $firma$, de un mensaje, m, es correcta, para ello utiliza las claves públicas del usuario y el número de variables de aceite, se va comprobando una a una si se cumple la igualdad 1.13, código 5.19.

Tipo	Nombre	Variable	Descripción
int vector	hashed	Input	Vector *hash*
int vector	firma	Input	Vector con la firma del *hash*
int vector	alpha_F2	Input	Vector de matrices triangulares superiores en $\mathbb{F}_2$, parte de la clave privada
int vector	beta_F2	Input	Matriz en $\mathbb{F}_2$, parte de la clave privada
int	m	Input	Número aceites
int vector	alpha	Inner	Almecena los correspondientes valores de alpha_F2 en el cuerpo de 128 elementos
int vector	beta	Inner	Almecena los correspondientes valores de beta_F2 en el cuerpo de 128 elementos
boolean	verif	Output	`True` si la firma es válida `False` en otro caso

Tabla 5.17: Parámetros de la función `verify`

```
def verify(hashed, firma, alpha_pub_F2, beta_pub_F2, m):
```

```

    verif = True
    alpha_pub = matrix3d_F2to128(alpha_pub_F2)
    beta_pub = matrix3d_F2to128(beta_pub_F2)
    for k in range(m):
        aux_alpha = matrix_product(matrix_product ([firma], alpha_pub[k]),
     matrix_transpose([firma]))
        aux_beta = matrix_product (beta_pub[k], matrix_transpose([firma]))
        verif = verif and (hashed[k] == matrix_sum(aux_alpha,  aux_beta)
     [0][0])

    return verif
```

Código 5.19: Verificación de la firma

Capítulo 6

Evaluación y pruebas

En este capítulo se debe proporcionar una medida objetiva de las bondades y beneficios de la solución propuesta, tanto en términos absolutos, como –en la medida de lo posible- comparándola con otras soluciones. Dependiendo del tipo de proyecto, debe incluir los resultados experimentales obtenidos al probar la solución; también puede incluir una tabla o diagrama de los costes reales del desarrollo, para así establecer conclusiones respecto a la planificación y costes estimados a priori. Finalmente, cuando se trata del desarrollo de una aplicación software, se pueden definir baterías de pruebas a realizar, de modo que en este capítulo se especificarán qué pruebas se han realizado, los resultados esperados y los resultados obtenidos.

Capítulo 7

Conclusiones

Capítulo en el que deben resumirse las principales aportaciones del trabajo realizado.

7.1. Valoración personal

Se puede incluir una valoración personal del proyecto (opcionalmente)

Bibliografía

[1] "Qilimanjaro: Next computing generation, quantum computation at your fingertips," 2018, https://neironix.io/documents/whitepaper/4514/whitepaper.pdf.

[2] Wikipedia, "Esfera de Bloch," https://es.wikipedia.org/wiki/Esfera_de_Bloch.

[3] A. Palau, "Tokenización & Árbol de Merkle," 2018, https://medium.com/@albpalau/tokenizaci%C3%B3n-%C3%A1rbol-de-merkle-1276820a1d60.

[4] L. Wilson, "The Rise of Quantum Computers – The Current State of Cryptographic Affairs," 2016, https://www.activecyber.net/rise-quantum-computers-current-state-cryptographic-affairs/.

[5] J. V. of Jeremiah Owyang, "Roadmap: Five Phases of Digital Eras," 2019, https://web-strategist.com/blog/2019/01/04/roadmap-five-phases-of-digital-eras/.

[6] Wikipedia, "Seguridad de la información," https://es.wikipedia.org/wiki/Seguridad_de_la_informaci%C3%B3n.

[7] Wikipedia, "Criptografía," https://es.wikipedia.org/wiki/Criptograf%C3%ADa.

[8] Wikipedia, "Criptografía Asimétrica," https://es.wikipedia.org/wiki/Criptograf%C3%ADa_asim%C3%A9trica.

[9] Wikipedia, "Atáque de fuerza bruta," https://es.wikipedia.org/wiki/Ataque_de_fuerza_bruta.

[10] Wikipedia, "Security Level," https://en.wikipedia.org/wiki/Security_level.

[11] Wikipedia, "Criptografía postcuántica," https://es.wikipedia.org/wiki/Criptograf%C3%ADa_postcu%C3%A1ntica.

[12] Wikipedia, "Restaurante Fogo de Chão," https://en.wikipedia.org/wiki/Fogo_de_Ch%C3%A3o.

[13] A. Szepieniec, B. Preneel, F. Vercaunteren, and W. Beullens, “LUOV: Signature Scheme proposal for NIST PQC Project (Round 2 version),” 2018, https://github.com/WardBeullens/LUOV/blob/master/Supporting_Documentation/luov.pdf.

[14] “Ark,” https://ark.io/.

[15] Wikipedia, “Computación cuántica,” https://es.wikipedia.org/wiki/Computaci%C3%B3n_cu%C3%A1ntica.

[16] Wikipedia, “Mecánica cuántica,” https://es.wikipedia.org/wiki/Mec%C3%A1nica_cu%C3%A1ntica.

[17] Álvaro Rodrigo Reyes, “Estado de la criptografía post-cuántica y simulaciones de algoritmos post-cuánticos,” http://openaccess.uoc.edu/webapps/o2/bitstream/10609/89026/6/alvaroreyesTFM1218memoria.pdf.

[18] Redacción Vivir, “La “nueva frontera” alcanzada por el computador cuántico de Google esta semana,” 08 septiembre 2020, https://www.elespectador.com/noticias/ciencia/la-nueva-frontera-de-la-computacion-cuantica/.

[19] A. Muñoz and J. I. Escribano, “La computación cuántica y el "futuro de la criptografía": la criptografía post-cuántica,” 2020, https://www.bbvanexttechnologies.com/la-computacion-cuantica-y-el-futuro-de-la-criptografia-la-criptografia-post-cuantica/.

[20] A. Banafa, “Computación cuántica y blockchain, mitos y realidades,” 2019, https://www.bbvaopenmind.com/tecnologia/mundo-digital/computacion-cuantica-y-blockchain-mitos-y-realidades/.

[21] Wikipedia, “Árboles Merkle,” https://es.wikipedia.org/wiki/%C3%81rbol_de_Merkle.

[22] “History of Blockchain,” https://academy.binance.com/blockchain/history-of-blockchain.

[23] “A brief history of Blockchain,” https://www.icaew.com/technical/technology/blockchain/blockchain-articles/what-is-blockchain/history.

[24] Wikipedia, “Reusable Proof of Work,” https://es.wikipedia.org/wiki/Reusable_Proof_Of_Work.

[25] “Contrato inteligente,” https://elderecho.com/los-contratos-inteligentes-smart-contracts-contratos-inteligentes.

[26] C. Pastorino, “Blockchain: qué es, cómo funciona y cómo se está usando en el mercado,” 2018, https://www.welivesecurity.com/la-es/2018/09/04/blockchain-que-es-como-funciona-y-como-se-esta-usando-en-el-mercado/.

[27] A. Rosic, "What is blockchain Technology? A Step-by-Step Guide For Beginners," https://blockgeeks.com/guides/what-is-blockchain-technology/#The_Three_Pillars_of_Blockchain_Technology.

[28] "Install docker engine on ubuntu," https://docs.docker.com/engine/install/ubuntu/.

[29] Github, "Versiones para descargar de ARK Wallet," https://github.com/ArkEcosystem/desktop-wallet/releases.

Siglas

DSA Digital Signature Algorithm.

ECDSA Elliptic Curve Digital Signature Algorithm.

RSA Rivest, Shamir y Adleman.

UOV Unbalanced Oil and Vinegar.

Apéndice A

Manual de usuario

El manual de usuario se va a realizar para una máquina ubuntu.

A.1. Instalación de Docker

En primer lugar instalamos la última versión de docker, [28].
Desintalar cualquier versión anterior de Docker.

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

Código A.1: Instalación Docker. Parte I

Actualizar los paquetes `apt` para tener acceso a las últimas actualizaciones e instalar los paquetes que permiten al sistema operativo acceder a los repositorios de Docker a través de HTTPS.

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates curl gnupg-
    agent software-properties-common
```

Código A.2: Instalación Docker. Parte II

Añadir la clave GPG oficial de Docker, la clave GPG es una característica de seguridad para asegurar que el software que se va instalar es auténtico.

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
    add -

OK
```

Código A.3: Instalación Docker. Parte III

Verificar que obtenemos la clave con la siguiente huella, para ello buscamos la huella con los últimos 8 dígitos, de la misma.

```
9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
```

```
$ sudo apt-key fingerprint 0EBFCD88

pub   rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid           [ unknown] Docker Release (CE deb) <docker@docker.com>
sub   rsa4096 2017-02-22 [S]
```

Código A.4: Instalación Docker. Parte IV

Instalar el repositorio de Docker.

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/
    linux/ubuntu $(lsb_release -cs) stable"
```

Código A.5: Instalación Docker. Parte V

Actualizar los repositorios que se acaban de agregar e instalar la última versión de Docker Engine y Docker Containerd.

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Código A.6: Instalación Docker. Parte VI

Verificar que se ha instalado correctamente comprobando la versión de Docker.

```
$ docker --version

Docker version 19.03.13, build 4484c46d9d
```

Código A.7: Instalación Docker. Parte VII

Algunos comandos útiles para el trabajo con Docker se muestran en el código A.8.

```
#Muestra los contenedores
$ sudo docker ps

#Lista los contenedores con los IDs
$ sudo docker container ls --all

#Lista las imagenes con los IDs
$ sudo docker images ls --all

#Guarda los cambios del docker
$ sudo docker commit <ID-CONTAINER> <NOMBRE-NUEVO:ETIQUETA>

#Corre un contenedor, abriendo los puertos indicados
$ sudo docker run -it -p <PUERTO:PUERTO> <NOMBRE:ETIQUETA>

#Elimina un contenedor
$ sudo docker rm <ID-CONTAINER>
```

Código A.8: Comandos útiles de Docker

A.2. Instalación Blockchain ARK

En Docker, iniciamos una imagen de ubuntu xenial. Además abrimos los puertos que van a ser necesarios posteriormente 4103, para la API pública, 4102, conexión P2P API paray 4200, para el *explorer*.

```
$ sudo docker run -ti -p 4103:4103 -p 4200:4200 ubuntu:xenial
```

Código A.9: Instalación *blockchain*. Parte I

Una vez estamos dentro de la máquina Docker instalamos sudo, para poder trabajar en modo administrador desde el usuario que vamos a crear.

```
$ apt-get install sudo
$ adduser deployer

Añadiendo el usuario `deployer' ...
Añadiendo el nuevo grupo `deployer' (1001) ...
Añadiendo el nuevo usuario `deployer' (1001) con grupo `deployer' ...
Creando el directorio personal `/home/deployer' ...
Copiando los ficheros desde `/etc/skel' ...
Introduzca la nueva contraseña de UNIX: ********
Vuelva a escribir la nueva contraseña de UNIX: ********
passwd: contraseña actualizada correctamente
Cambiando la información de usuario para deployer
Introduzca el nuevo valor, o presione INTRO para el predeterminado
  Nombre completo []:
  Número de habitación []:
  Teléfono del trabajo []:
  Teléfono de casa []:
  Otro []:
¿Es correcta la información? [S/n] S
```

Código A.10: Instalación *blockchain*. Parte II

Cambiar el modo del usuario deployer incluyéndolo en el grupo sudo para que sea un superusuario y finalmente, entrar al usuario deployer.

```
$ usermod -aG sudo deployer
$ su - deployer
```

Código A.11: Instalación *blockchain*. Parte III

Actualizamos los paquete e instalamos algunos nuevos como `git`, `curl` y `yarn`.

```
$ sudo apt-get update
$ sudo apt-get install git curl yarn jq apt-transport-https
```

Código A.12: Instalación *blockchain*. Parte IV

Instalar las dependencias nvm.

```
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.8/
    install.sh | bash
```

Código A.13: Instalación *blockchain*. Parte V

Para comprobar que se ha instalado correctamente nos salimos del usuario deployer y volvemos a entrar.

```
$ command -v nvm

nvm
```

Código A.14: Instalación *blockchain*. Parte VI

Para eliminar nvm

```
$ nvm use system
$ npm uninstall -g a_module
$ sudo npm  install -g npm
```

Código A.15: Instalación *blockchain*. Parte VII

Instalar pm2.

```
$ sudo apt-get install npm
$ sudo npm i -g pm2
$ ln -s /usr/bin/nodejs /usr/bin/node
```

Código A.16: Instalación *blockchain*. Parte VIII

Comando interesantes para trabajar con pm2. Nos servirar para observar si la *blockchain* y el *explorer* están levantados.

```
#Lista los demonios de pm2
$ pm2 list

#Se obtienen los estados de los demonios de pm2
$ pm2 status
```

Código A.17: Comandos útiles de pm2

Hay que clonar el directorio deployer de `@ArkEcosystem`.

```
$ git clone https://github.com/ArkEcosystem/deployer.git
$ chmod 764 deployer/setup.sh
$ ./setup.sh
```

Código A.18: Instalación *blockchain*. Parte IX

Iniciar la base de datos e instalar tanto la *blockchain* como el *explorer*. Una vez que instalemos el `core` obtendremos en la salida la dirección y la *passphrase* del wallet Genesis la tenemos que guardar en un archivo para más tarde poder hacer las transacciones, aún así podemos encontrar la *passphrase* junto con la dirección en el archivo `/home/<USUARIO>/.bridgechain/testenet/<NOMBRE-BRIDGECHAIN>/genesisWallet.json`. Esto puede tardar unos minutos.

```
$ sudo service postgresql start
$ sudo apt-get update -y
$ sudo apt-get install -y libjemalloc-dev
$ ./deployer/bridgechain.sh install-core --config deployer/config.sample.
    conf --autoinstall-deps --non-interactive
```

Código A.19: Instalación *blockchain*. Parte X

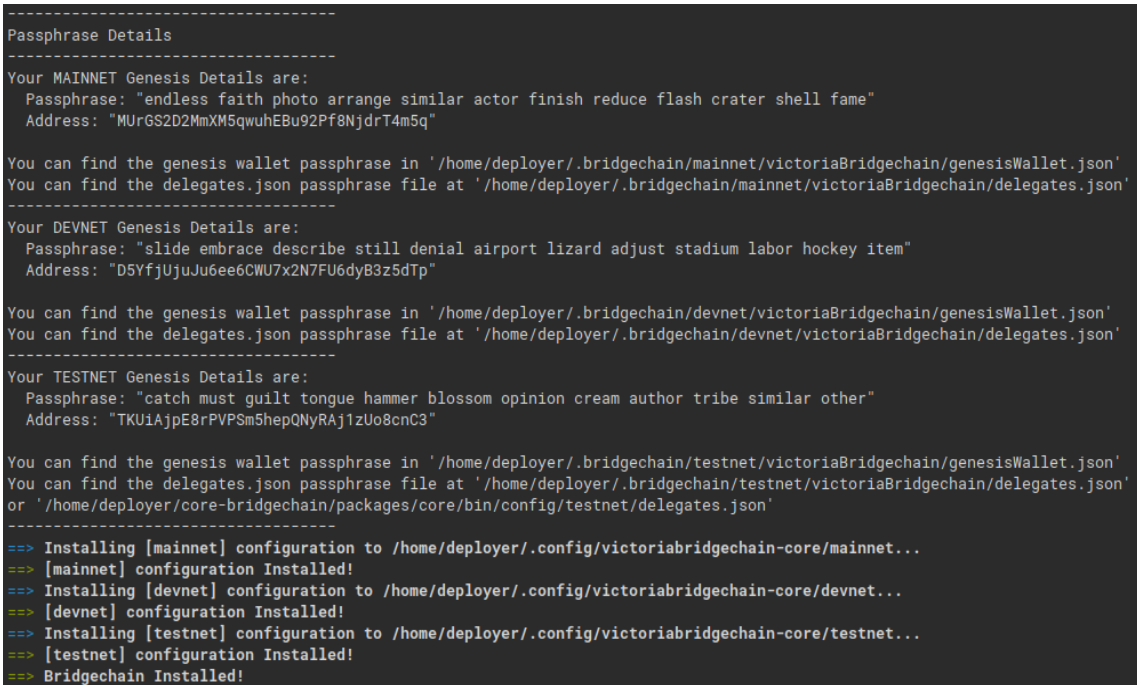

Figura A.1: Salida tras la instalación del core

```
$ ./deployer/bridgechain.sh install-explorer --config deployer/config.
    sample.conf --skip-deps --non-interactive
```

Código A.20: Instalación *blockchain*. Parte X

Iniciamos la *blockchain* y el *explorer*.

```
$ ./deployer/bridgechain.sh start-core --network testnet

==> Starting...
Starting victoriabridgechain-relay... done
Starting victoriabridgechain-forger... done
==> Start OK!


$ ./deployer/bridgechain.sh start-explorer --network testnet
```

Código A.21: Instalación *blockchain*. Parte XII

La imagen A.2 muestra la salida después de haber iniciado el *explorer*. Ese mismo cuadro es el que se visualiza tras ejecutar `pm2 status` pues muestra tanto si la *blockchain* como el *explorer* están iniciados.

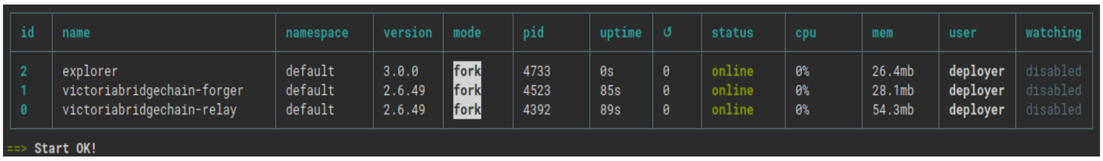

Figura A.2: Salida tras iniciar del *explorer*

Para visualizar las transacciones en el *explorer*, debemos de abrir un navegador y acceder a la url `http://NODE_GENESIS_IP:EXPLORER_PORT`, donde el `EXPLORER_PORT` es 4200. La imagen A.3 muestra las primeras transacciones realizadas, entre ellas se encuentra la transacción inicial al monedero génesis, además del registro de los delegados.

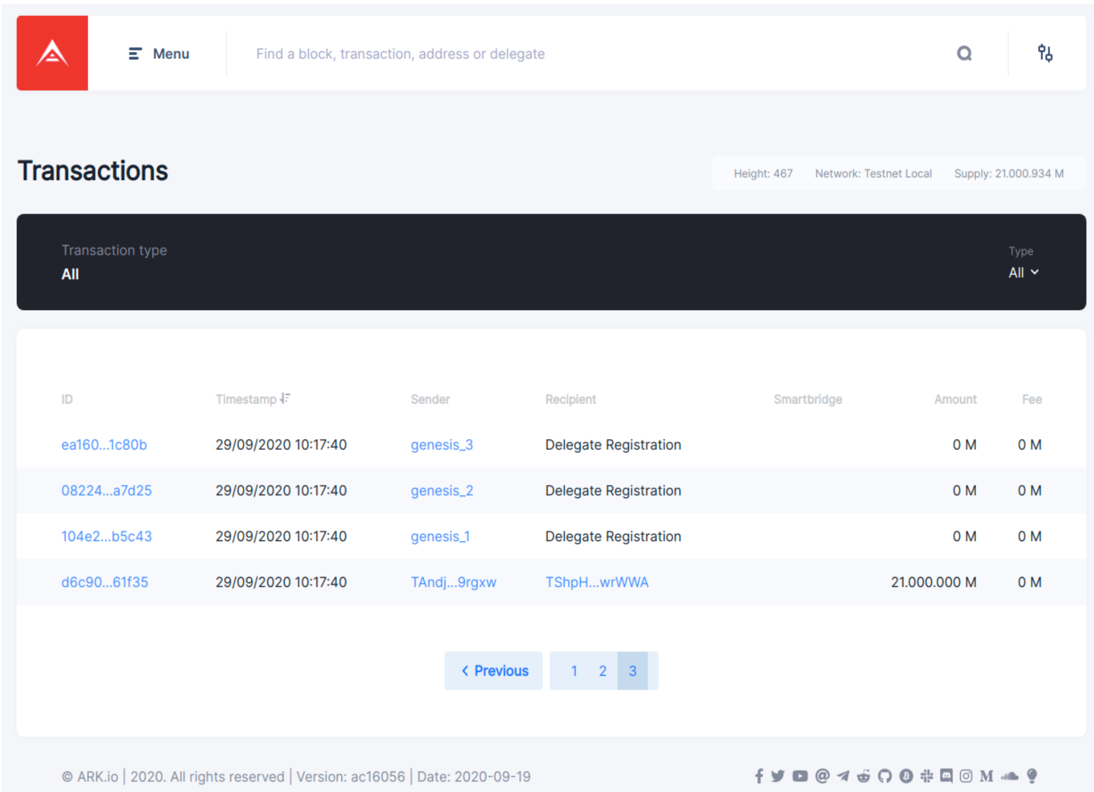

Figura A.3: Primeras transacciones en el *explorer*

A.3. Instalación de la aplicación ARK Desktop Wallet

Antes de descargar la aplicación ARK Desktop Wallet, en el ordenador local, son necesarias algunas instalaciones previas, como algunos archivos de desarrollo de `libudev`, `node 12` y `yarn`, código A.23.

```
$ sudo apt-get install libudev-dev libusb-1.0-0-dev

$ npm install -g n
$ sudo n 12

# Para comprobar que se ha instalado node 12
$ n --version

$ npm install -g yarn
```

Código A.22: Instalaciones previas a la aplicación ARK Wallet

La descarga de la aplicación se realiza desde el repositorio de github de `@ArkEcosystem/desktop-wallet` [29], para el proyecto se ha descargado la última versión del archivo `.deb`.

```
$ sudo dpkg -i ark-desktop-wallet-linux-amd64-<VERSION>.deb

#Para desintalarlo
$ sudo apt-get remove ark-desktop-wallet
```

Código A.23: Instalación de la aplicación ARK Wallet

Una vez que se ha instalado la aplicación, la abrimos y nos encontramos con la imagen A.4. A continuación, vamos siguiendo los pasos que nos salen en la parte de la derecha de la pantalla.

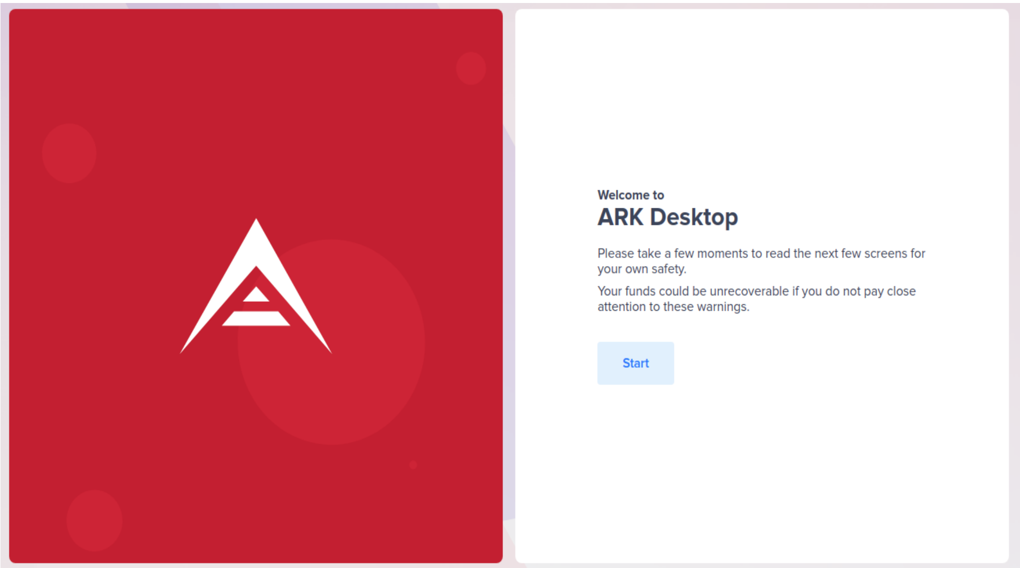

Figura A.4: Inicio de Wallet

Así se llega al primer paso, crear un perfil, en el que hay que indicar el nombre y la moneda con la que se quiere trabajar, en este caso se dejan los valores por defecto, además existe la opción de cambiar el avatar del usuario, imagen A.5.

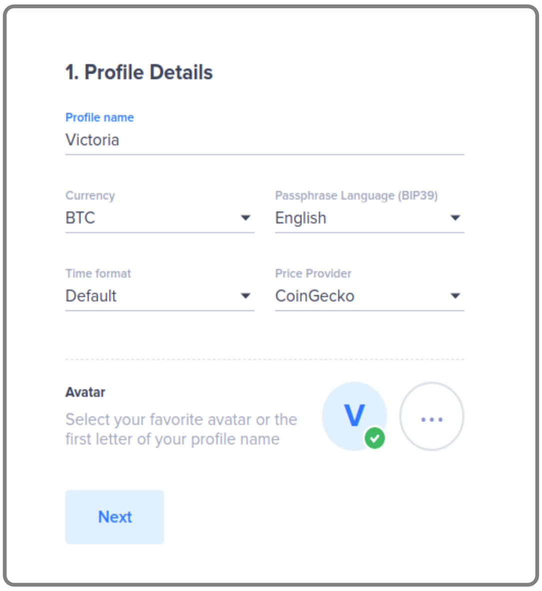

Figura A.5: Detalles del perfil

En el siguiente paso aparece un menú con las posibles redes, *mainnet* y *devnet*. Sin embargo, si se desea usar la red *testnet* es necesario crearla pulsando nueva red. Los campos a rellenar son el nombre, una breve descripción y la dirección del servidor con el puerto de la API, `http:/GENESIS_NODE_IP:API_PORT`, imagen A.6.

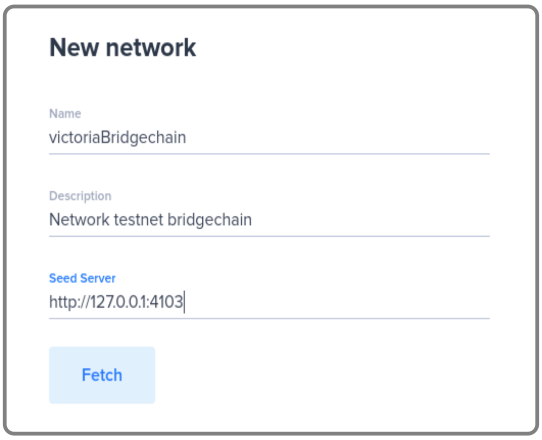

Figura A.6: Configuración de la nueva red

Cuando se ha creado la red aparece una pantalla con los detalles de la misma donde debemos de cambiar la dirección por defecto del explorer y poner `http://GENESIS_NODE_IP:EXPLORER_PORT`. También se pueden cambiar el nombre de los Tokens y el símbolo. Una vez realizados los cambios, guardamos la red, imagen A.7.

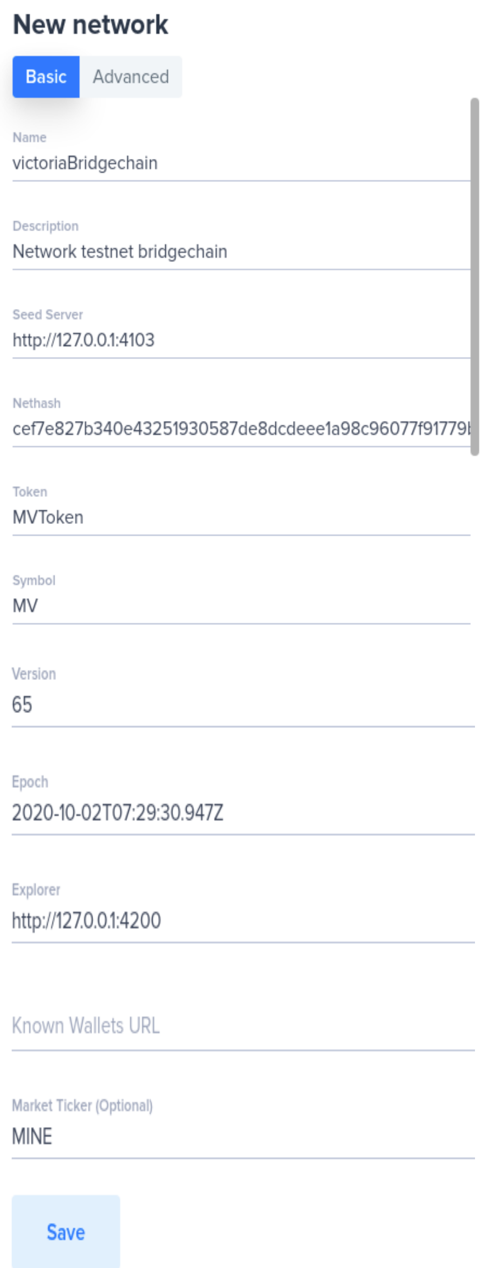

Figura A.7: Detalles de la nueva red

Se selecciona la red recien creada, `victoriaBridgechain` y se avanza al siguiente paso, imagen A.8.

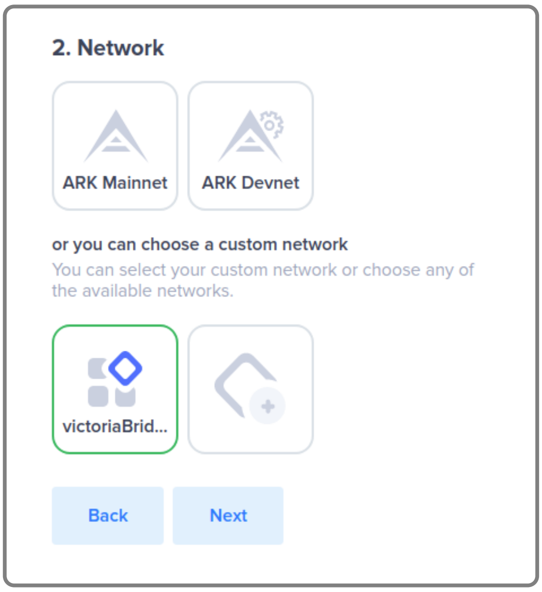

Figura A.8: Selección de la red

Para acabar con la creación del usuario, se tiene la opción de personalizar los colores de la interfaz de usuario o cambiar al tema de noche, imagen A.9.

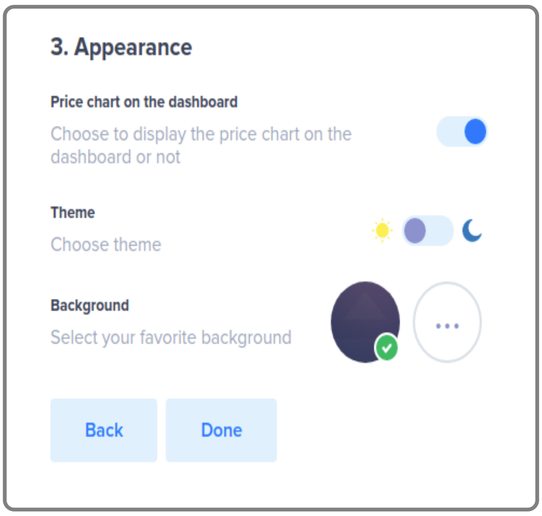

Figura A.9: Personalización del diseño de la interfaz

La primera vez que se entra al usuario es necesario importar el monedero que se ha creado durante la instalación de la *blockchain*, pulsando en la esquina superior derecha en `Import Wallet`. No es necesario rellenar los dos campos, se puede importar el monedero introduciendo solamente la *passphrase*, imagen A.10.

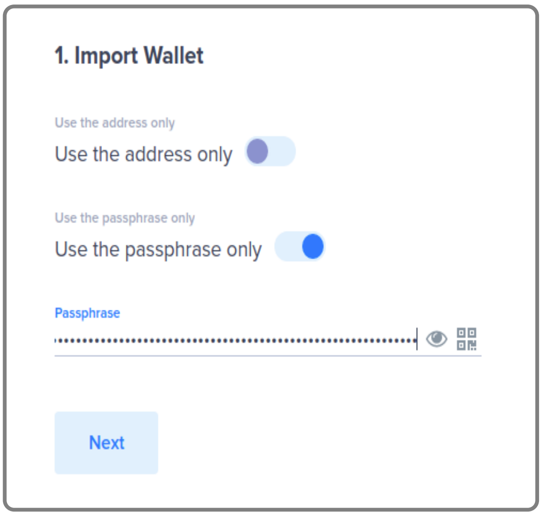

Figura A.10: Importar monedero

Si se desea se puede poner contraseña al monedero, haciéndolo más seguro, en el ejemplo no se van a poner contraseña a ninguno de los monederos, imagen A.11.

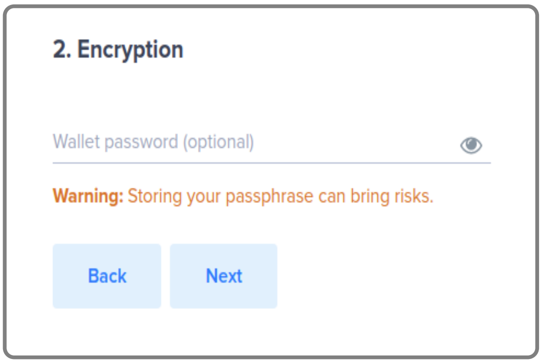

Figura A.11: Encriptación el monedero

Finalmente, hay que ponerle nombre al monedero, para diferenciarlo de otros monederos y saber cual es el que contiene la cantidad inicial, el nombre que se le va a poner es `MainWallet`, imagen A.12.

Figura A.12: Confirmación para crear el monedero

Para que salga la cantidad de dinero inicial es necesario conectar la aplicación con la *blockchain*. Pulsando en el panel lateral izquierdo en `Network`, se accede a la configuración de la red y pulsando en `Connect custom peer` obtenemos la imagen A.13. Rellenamos los campos con `http://GENESIS_NODE_IP` y con el `API_PORT`. Cuando se conecte hay que refrescar la página para que se actualice el monedero con el dinero.

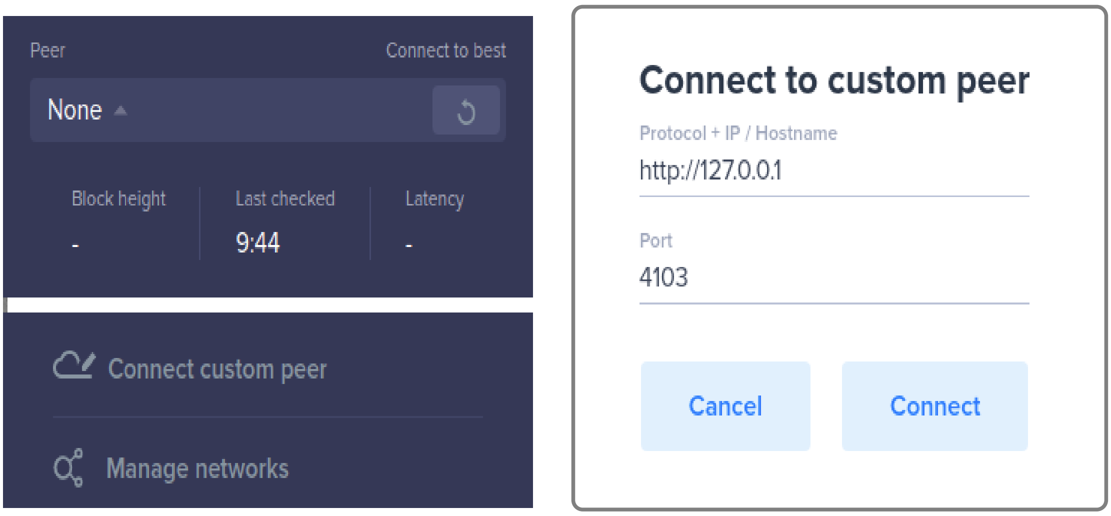

Figura A.13: Configuración de la conexión peer

Para realizar transacciones y poder probar el algoritmo, es necesario crear otro monedero y realizar transacciones entre ambos. De esta forma, hay que acceder a la sección `My wallets` y pulsar `Create Wallet`. El primer paso es introducir el nombre del monedero, imagen A.14.

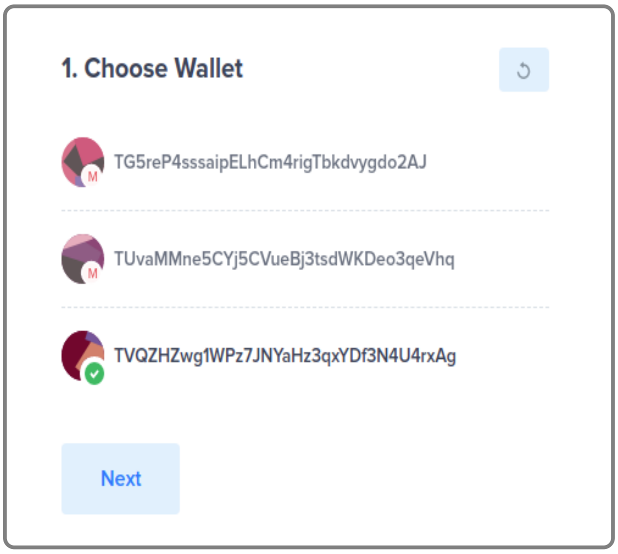

Figura A.14: Selección de la dirección del nuevo monedero

La imagen A.15 muestra la *passphrase* del monedero. Esta *passphrase* hay que guardarla, tal y como se hizo con la *passphrase* de monedero importado, pues será necesaria para realizar transacciones posteriormente.

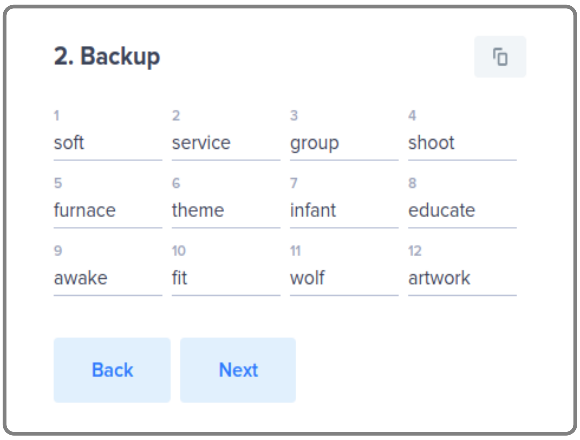

Figura A.15: *Passphrase* o clave privada del monedero

La verificación de la *passphrase* se hace introduciendo tres palabras, la número tres, seis y nueve, si se desea se puede realizar la verifiación introduciendo todas las palabras, imagen A.16.

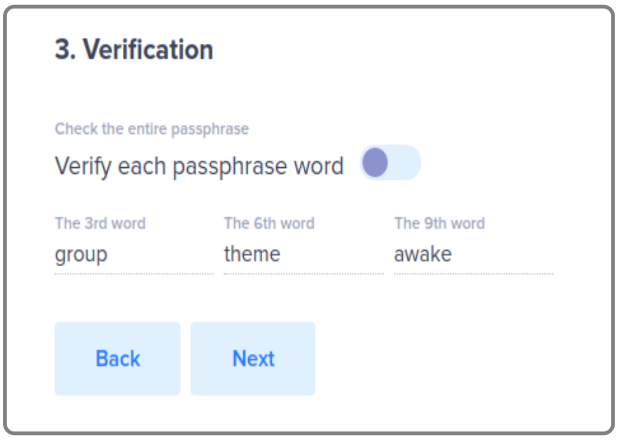

Figura A.16: Verificación de la *passphrase*

Si se desea se puede poner contraseña al monedero, en el ejemplo no es necesario, imagen A.17.

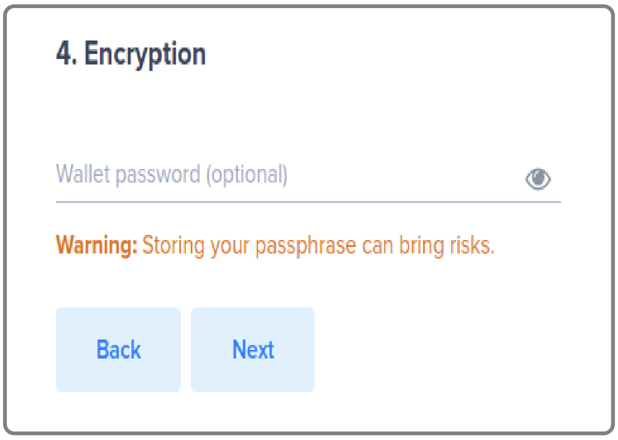

Figura A.17: Encriptación del monedero

Antes de confirmar la creación del monedero, hay que introducir el nombre del mismo, `Second Wallet`, para hacer referencia a que no tendrá ningún token hasta que no reciba una transferencia, imagen A.18.

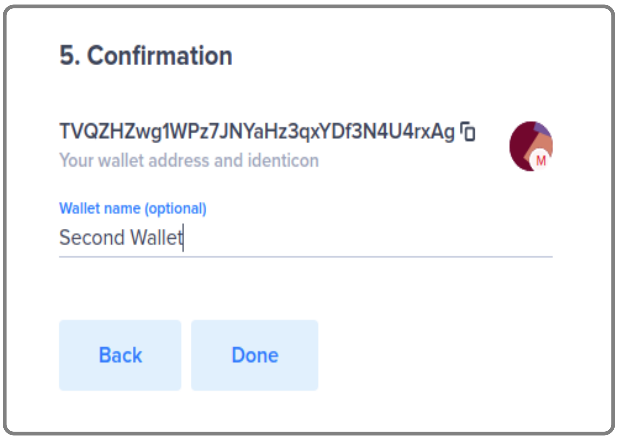

Figura A.18: Confirmación para crear el segundo monedero