Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Implementación de un sistema de comunicaciones seguro mediante Blockchain y su aplicación al Internet de las Cosas.

Autor: Daniel Ferreiro Rodríguez

Tutor: José Manuel Lanza Gutiérrez

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

Trabajo Fin de Grado

Implementación de un sistema de comunicaciones seguro mediante Blockchain y su aplicación al Internet de las Cosas.

Autor: Daniel Ferreiro Rodríguez

Tutor: José Manuel Lanza Gutiérrez

Tribunal:

Presidente: Nombre del presidente del tribunal

Vocal 1º: Nombre del primer vocal

Vocal 2º: Nombre del segundo vocal

FECHA: 8 de septiembre de 2022

"Lo que sabemos es una gota de agua; lo que ignoramos es el océano."

Isaac Newton

Agradecimientos

El mundo de la Informática siempre me llamó la atención desde muy pequeño suscitado por los videojuegos y mi madre.

No ha sido un camino fácil para aquel adolescente, que con tan solo 16 años, se abría paso en un país extranjero. Para entonces, conocía el Internet por lo que había leído en los libros. Ha sido un proceso de mucho aprendizaje y evolución tecnológica que empezó con el estudio del curso de FP en Desarrollo de Aplicaciones Multiplataforma (DAM). En este curso desarrollé habilidades, adquirí conocimientos y generé aún más, si cabe, el amor hacia esta profesión. Una vez graduado, renuncié a una oferta de trabajo y estudié para la fase específica de la Selectividad con el objetivo de hacerme de una plaza en la Universidad. Lo conseguí. En esta nueva etapa estudiantil, conocí personas maravillosas que me han hecho mejor profesional, pero sin duda, mejor persona. Con este trabajo concluyo mi carrera, logrando mi propósito y deseo, ser ingeniero informático.

Quiero agradecer a mi familia, más concretamente a mis padres y mi hermano, quienes me han dado su confianza, apoyo y fuerza en todo lo que he hecho en la vida. Por alegrarse, más que yo, por mis logros y estar conmigo en los momentos difíciles, en los días no tan buenos. Por ser como son y convertirme en la persona que soy hoy en día. Este título, también es de ustedes.

A mi tutor del Trabajo Final de Grado, José Manuel, que siempre ha tenido un espacio para resolver todas mis dudas, me ha acompañado y guiado de forma exquisita en el último paso de mi carrera.

La carrera me permitió dar una nueva oportunidad al amor al conocer a Ana. Con la que he compartido clases, trabajo, largas noches de estudio y un sentimiento. En varias fases del proyecto, fue una fuente de inspiración para mí y la he bautizado como “la musa del *Blockchain*”. Muchas gracias por escuchar mis razonamientos, aunque le “sonaran a chino”, por su tiempo, ayuda y su saber estar.

Quiero agradecer a mis amigos para toda la vida: Fran, Sebas, Nacho, Emi, Raúl, Cosmin... y a todas las personas que de una forma directa o indirecta me han ayudado a conseguir esta meta.

Muchísimas gracias por todo.

Resumen

En este proyecto se analizan las posibilidades que brinda la tecnología *Blockchain* para el envío de información sensible de forma segura dentro de la arquitectura del Internet de las Cosas. Para ello, se ha optado por desarrollar una solución que consta de las tres capas habituales en el Internet de las Cosas: *edge*, *fog* y *cloud*. En el *edge* se ha implementado un dispositivo ESP32 que genera cadenas aleatorias para ser enviadas al *fog*, una Raspberry Pi en este caso. La Raspberry cifra la información mediante el algoritmo AES y la envía hacia el *cloud*, previo encadenamiento en un *Blockchain* para incrementar la seguridad y confiabilidad de la comunicación. En el *cloud*, una máquina virtual de Azure, se recibe y comprueba la validez de cada mensaje recibido del *fog*, para luego ser guardados en una base de datos no relacional, *MongoDB*, una vez desencriptado.

Bajo esta solución implementada se ha llevado a cabo un estudio experimental donde se ha ido variando el tamaño del mensaje enviado, obteniendo diversas estadísticas, como el tiempo de procesado o la memoria RAM usada. Como fruto de este estudio, se puede concluir que es viable asegurar las comunicaciones en el Internet de las Cosas, en este caso concreto entre *fog* y *cloud*, pero que se debe poner especial atención al tamaño del mensaje enviado, ya que influye negativamente en todos los parámetros analizados.

Palabras clave: *Blockchain*, IoT, *MongoDB*, AES.

Abstract

This project analyzes the possibilities offered by Blockchain technology for sending sensitive information safely within the architecture of the Internet of Things. To do this, it has been decided to develop a solution that consists of the three usual layers on the Internet of Things: edge, fog and cloud. At the edge, an ESP32 device has been implemented that generates random strings to be sent to the fog, a Raspberry Pi in this case. The Raspberry encrypts the information using the AES algorithm and sends it to the cloud, after chaining it in a Blockchain to increase the security and reliability of communication. In the cloud, an Azure virtual machine, receives and checks the validity of each message received from the fog, to then be saved in a non-relational database, MongoDB, once decrypted.

Under this implemented solution, an experimental study has been carried out where the size of the message sent has been varied, obtaining various statistics, such as the processing time or the RAM used. As a result of this study, it can be concluded that it is feasible to secure communications on the Internet of Things, in this specific case between fog and cloud, but special attention must be paid to the size of the message sent, since it has a negative influence on all the analyzed parameters.

Keywords: Blockchain, IoT, MongoDB, AES.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos principales

IoT	Internet of Things
WiFi	Wireless Fidelity
JSON	JavaScript Object Notation
ASES	Advanced Encryption Standard
IDE	Integrated Development Environment
SQL	Structured Query Language
SSH	Secure SHell
MV	Máquina Virtual
BD	Base de Datos
TCP	Transmission Control Protocol

1 Introducción

Este apartado tiene como objetivo introducir al lector en la motivación que llevó a realizar el proyecto. Además, se definen los objetivos de éste, la solución propuesta, y por último, se detalla la estructura de la memoria.

1.1 Motivación

El proyecto que se presenta en esta memoria va dirigido al desarrollo de un sistema informático que emplee la tecnología ***Blockchain*** [1] para asegurar las comunicaciones, especialmente aquellas de tipología sensible, de dispositivos extremos del **Internet de las Cosas** (IoT, del término anglosajón *Internet of Things*) [2]. El ámbito de aplicación de este trabajo se encuentra bajo el paraguas de un proyecto de investigación que busca prevenir y combatir la violencia de género mediante una solución tecnológica basada en IoT.

El ***Blockchain*** **(**Cadenas de bloques en español**)** es un conjunto de tecnologías que combinadas hacen posible que los dispositivos puedan gestionar su información compartiendo un registro distribuido**,** descentralizado y sincronizado entre todos ellos, en vez de utilizar las bases de datos tradicionales. La información se transmite y guarda de un modo extremadamente seguro, respetando la identidad **y** privacidad**,** gracias al uso de claves criptográficas. Los registros no pueden ser alterados, borrados ni sobrescritos una vez registrados. Son visibles para cualquier participante de la red (si ésta es pública), añadiendo una gran transparencia.

El IoT es una tecnología en pleno desarrollo que permite integrar el mundo físico con los sistemas informáticos, permitiendo la recolección e intercambio de datos a través de Internet. Esta tecnología se estructura habitualmente en torno a tres niveles: *edge* (extremo), *fog* (niebla) y *cloud* (nube). En el *edge* se encuentran aquellos dispositivos que están desplegados de forma muy cercana al usuario y cuyo cometido es monitorizar su entorno. La característica principal de los dispositivos en esta capa es su restricción en capacidad de cómputo y almacenamiento, con el propósito de reducir su tamaño, así como la disipación de energía, mientras que su gestión energética se maximiza (se aumenta su autonomía en el habitual supuesto de requerir de baterías). En el *fog* se encuentran aquellos dispositivos que proporcionan una capacidad de cómputo sustancialmente superior a los del *edge*, pero sustancialmente inferior a los del *cloud*, a cambio de encontrarse más cerca del *edge*. En el *cloud* se encuentran los grandes centros de cálculo, los cuales se encuentran a gran distancia del *edge* y requieren de ingentes cantidades de energía para su funcionamiento, a cambio de unas excelentes capacidades computacionales.

Como se ha introducido previamente, el ámbito final de aplicación se centra en ayudar a combatir la violencia de género. La discriminación hacia las mujeres y la violencia de género es un problema que traspasa fronteras y que está presente lamentablemente en la mayor parte de los países. Las personas LGBT+ y las mujeres sufren violencia por el mero hecho de serlo, reafirmándose como una privación de los derechos humanos y las libertades fundamentales de la sociedad. En este sentido, la Organización de las Naciones Unidas [3] dijo que el término violencia de género se utiliza “para distinguir la violencia común de aquella que se dirige a individuos o grupos sobre la base de su género”.

Como resultado de esta problemática, la mayor parte de los gobiernos invierten en soluciones tecnológicas que ayuden a identificar casos de esta tipología con la mayor celeridad posible. Por ejemplo, en España, las fuerzas de seguridad hacen uso de un dispositivo denominado **COMETA**, el cual es entregado a víctimas de especial protección y consta de un localizador GPS y un pulsador que debe ser activado en caso de riesgo [4]. Sin embargo, esta solución presenta dos problemas. El primero es que solamente llega a víctimas de especial relevancia, por lo que su penetración en la sociedad es muy limitada. El segundo es que requiere que la víctima pulse el botón de auxilio, lo cual es muchas ocasiones no es posible, ya sea porque no se lo permiten o bien, por un bloqueo momentáneo debido al estrés de la situación.

En base a las limitaciones de este tipo de soluciones surge **EMPATIA-CM**, el cual es un proyecto liderado por un diverso equipo formado por grupos de investigación de diferentes universidades de la Comunidad de Madrid (Universidad Carlos III de Madrid [5], Universidad Politécnica de Madrid y Universidad de Alcalá), teniendo como meta la creación y desarrollo de un sistema portable llamado **Bindi** [6] que ayude a prevenir y combatir la violencia de género. Para ello, el sistema contará con un sistema de inteligencia que, partiendo de información física y fisiológica de la víctima o su entorno, será capaz de detectar y avisar automáticamente de situaciones de violencia de género, estando estas situaciones íntimamente ligadas a la detección de ataques de miedo o pánico.

Bindi está formado por los siguientes componentes principales:

- Pulsera: dispositivo del *edge* de IoT que se encarga de monitorizar distintas variables fisiológicas, como temperatura y conductividad de la piel, así como ritmo cardiaco. Incluye un sistema de inteligencia de primer nivel que se ejecuta de forma embebida en el dispositivo en base a la información fisiológica capturada.
- Colgante: dispositivo del *edge* de IoT que se encarga de capturar el audio del entorno y la voz de la víctima.
- Móvil: dispositivo del *fog* de IoT que se encarga de gestionar la información procedente de la pulsera y el colgante. Además, se encarga de ejecutar las acciones pertinentes en caso de detectar un caso de violencia de género en base a la información recolectada. Incluye un sistema de inteligencia de segundo nivel en base a la información fisiológica y física capturadas por la pulsera y el colgante, respectivamente.
- Servidor: sistema informático del *cloud* de IoT que se encarga de gestionar la ocurrencia de las alarmas de casos de violencia de género, así como de almacenar aquella información relevante capturada por los distintos dispositivos. Incluye el mantenimiento y encriptado de información susceptible de actuar como prueba judicial.

Como se puede observar, el tipo de información capturada por este sistema informático es extremadamente sensible, por el propio ámbito de la aplicación (posibles situaciones de peligro) y por ser información de tipo personal. Por ello, se requiere proteger esta información en el punto posible más cercano a su captura, ya sea en el *edge* (pulsera y colgante) o el *fog* (móvil), siendo especialmente desafiante llevar a cabo esta tarea en el *edge* debido principalmente a la inmadurez de esta tecnología y al tipo de *hardware* limitante del que suele hacerse uso, como se comentó previamente.

Con el propósito de abordar esta problemática, este proyecto plantea proporcionar soluciones que puedan ayudar a proteger la información de este sistema mediante el uso de la tecnología ***Blockchain*** y teniendo en cuenta la limitación impuesta por el *edge*. Para ello, se propone la construcción de ***Gateways* o interfaces de comunicación** especializadas, esto es, el uso del *fog* como punto de concentración de información cifrada para su envío al *cloud*. Como prueba de concepto sobre el uso de *Blockchain* para asegurar las comunicaciones IoT en Bindi, este proyecto construirá una arquitectura similar a la ya usada en Bindi, pero no hará uso de los dispositivos Bindi, sino que hará uso de dispositivos técnicamente similares. La motivación de esta decisión se centra en facilitar esta prueba de concepto, ya que los dispositivos en Bindi son prototipos y no dispositivos comerciales, como los usados aquí.

1.2 Objetivos

El objetivo principal de este proyecto es hacer un estudio investigativo con el propósito de identificar posibles soluciones que permitan utilizar la tecnología ***Blockchain*** en **IoT,** siguiendo para ello la estrategia de construcción de ***Gateways*** especializados.

Para ello, se requiere alcanzar una serie de subobjetivos:

- Avanzar en el conocimiento de la tecnología *Blockchain*.
- Identificar bibliotecas que permitan implementar la tecnología *Blockchain* en distintos tipos de

dispositivos.

- Construcción, verificación y caracterización de una solución basada en *Gateway*, que permita traducir la información no asegurada de un dispositivo, en información asegurada mediante *Blockchain*.

El principal campo de aplicación del estudio llevado a cabo en este proyecto está relacionado directamente con el proyecto EMPATIA-CM y su solución tecnológica Bindi. De esta manera, los estudios resultantes de este proyecto podrían ayudar a mejorar y paliar los problemas de seguridad en el *edge*/*fog* de IoT en el que se basa la solución. Especialmente en aquellas situaciones donde se requiere del envío de información física y fisiológica desde el *edge*/*fog* (pulsera, colgante y teléfono móvil) al *cloud*. Los resultados de este proyecto también podrían ser aplicados a otros ámbitos de aplicación donde se requiera hacer uso de tecnología del *edge* de IoT.

1.3 Solución propuesta

La solución propuesta en este proyecto consiste en un sistema que permite enviar información "sensible" del *edge*/*fog* al *cloud*. La seguridad de los datos se logra empleando la tecnología *Blockchain* de forma sencilla y completa en donde cada bloque tiene su propio hash[1] y es validado pertinentemente.

Se han llevado a cabo dos implementaciones, siguiendo una metodología incremental, donde la primera solución, cronológicamente hablando, se expande para llevar a cabo una segunda solución.

- Solución 1: Se hace uso de una Raspberry Pi que hace las funciones de dispositivo del *fog* (debido a sus características técnicas). Este dispositivo genera la información (cadenas de texto aleatorias) que enviará al *cloud*, una máquina virtual de Microsoft Azure, de forma asegurada mediante Blockchain.
- Solución 2: Se hace uso de la Raspberry Pi de la solución anterior, como dispositivo del *fog*, pero se incluye un dispositivo ESP32 que genera la información (cadenas de texto aleatorias) que es enviada al *fog*, para ser asegurada y enviada al *cloud*. Esta solución simula la arquitectura utilizada en Bindi.

A continuación, se introducen algunos detalles comunes a ambas soluciones:

- Los bloques de información enviados desde el *fog* son encadenados en *Blockchain* incluyendo los siguientes componentes: número de bloque, timestamp con la fecha y hora de creación del bloque, la información procedente del *edge*/*fog* cifrada por el algoritmo AES (acrónimo del término anglosajón *Advanced Encryption Standard*), hash del bloque, hash del bloque anterior y conjunto de variables estadísticas tomadas en el proceso de cifrado y descifrado de las cadenas.
- Los bloques de información (ya encadenados) son enviados desde la Raspberry Pi a la máquina virtual de Azure en forma de un fichero JSON (acrónimo del término anglosajón *JavaScript Object Notation*)[2], usando para SSH (acrónimo del término anglosajón *Secure SHell*)[3]. Cada bloque se muestra en la consola de la Raspberry Pi.
- En la máquina virtual (*cloud*), los bloques recibidos son validados para comprobar que la información no ha sufrido cambios en el envío. Se desencripta el mensaje del bloque y su contenido es mostrado por la consola del sistema operativo ejecutando en la máquina virtual. En el caso de que la información recibida sea correcta, ésta es almacenada en una base de datos no relacional *MongoDB*.

[1] Una función criptográfica hash- usualmente conocida como "hash"- es un algoritmo matemático que tiene como objetivo codificar datos para formar una cadena de caracteres única y de longitud fija. Esta cadena es habitualmente utilizada para comprobar la validez de la información recibida en una comunicación.

[2] Formato ligero de intercambio de datos, que resulta sencillo de leer y escribir para los programadores y simple de interpretar y generar para las máquinas.

[3] Nombre del protocolo y programa que lo implementa cuya principal función es el acceso remoto a un servidor por medio de un canal seguro en el que toda la información está cifrada.

1.4 Estructura de la memoria

Este documento cuenta con distintos capítulos en los que se recoge la información más relevante acerca del desarrollo llevado a cabo en este proyecto, así como los fundamentos teóricos requeridos para la comprensión de esta memoria. Cada uno de estos apartados se describe brevemente a continuación:

- **Capítulo 1. Introducción**. En este primer capítulo se introduce al lector en la motivación para la realización del proyecto, se definen los objetivos, se detalla la solución propuesta y la estructura de esta memoria.
- **Capítulo 2. Base teórica**. Se exponen los fundamentos teóricos requeridos para una correcta compresión de esta memoria.
- **Capítulo 3. Arquitecturas propuestas**. Se realiza una descripción de los distintos componentes que conforman el sistema, así como de la estructura del modelo de datos.
- **Capítulo 4. Implementación del sistema**. Se describen los detalles más relevantes acerca de la implementación en las tres plataformas: ESP32 (*edge*), Raspberry Pi (*fog*) y la máquina virtual de Azure (*cloud*).
- **Capítulo 5. Experimentación**. Se presenta un estudio experimental donde se analiza la influencia del tamaño de los datos enviados en el uso de esta tecnología basada en *Blockchain*.
- **Capítulo 6. Conclusiones y líneas futuras**: Se exponen las principales conclusiones obtenidas durante el desarrollo del sistema. Se analizan las líneas futuras de trabajo, incluyendo posibles mejoras del sistema.
- **Bibliografía**: Se lista la bibliografía utilizada para el desarrollo de este proyecto. El estilo usado para citar la bibliografía a lo largo de la memoria fue el IEEE (*Institute of Electrical and Electronics Engineers*).
- **Anexo I**: Se expone el presupuesto para el desarrollo del proyecto.

2 Base teórica

En este apartado se abordan los conceptos principales de las tecnologías empleadas para la realización de este proyecto. Se incluyen temas como el *Blockchain*, el IoT y el cifrado.

2.1 Blockchain

Blockchain es un libro mayor compartido [7] e inmutable que facilita el proceso de registro de transacciones y de seguimiento de activos en una red de negocios. Un activo puede ser tangible (una casa, un vehículo, dinero en efectivo, terrenos) o intangible (propiedad intelectual, patentes, derechos de autor, marcas). Prácticamente cualquier cosa de valor puede ser rastreada y comercializada en una red de *Blockchain*, reduciendo el riesgo y los costos para todos los involucrados.

La información que se almacena en dicha red dependerá del propósito para el que haya sido creada. Puede tratarse de una red que almacene datos de pago (moneda criptográfica o criptomonedas), información médica, datos logísticos o de trazabilidad de alimentos e inclusive recuento de datos electorales.

Debe remarcarse que ***Blockchain* no es una criptomoneda.** En 2008, Bitcoin, la primera y más exitosa criptomoneda, utilizó la tecnología *Blockchain* para crear una red en la cual personas anónimas pudiesen realizar transferencias de moneda no fiduciaria sin necesidad de intermediarios. Tras el nacimiento de Bitcoin comenzaron a lanzarse muchas otras criptomonedas utilizando *Blockchain*, siendo hoy ya más de 2.000. Más allá de las criptomonedas, el éxito de la tecnología *Blockchain* para generar consenso, funcionar de forma descentralizada y resistir intentos de hackeo con su condición de inmutabilidad, motivó a la comunidad emprendedora a considerarla como una opción atractiva para construir soluciones digitales a nivel gubernamental y empresarial [8].

2.1.1 Funcionamiento

Hay ciertos ingredientes que casi todos los *Blockchains* tienen en común por defecto. Los tres elementos clave en la función del *Blockchain* son sus participantes, activos, y transacciones.

2.1.1.1 Participantes

Los participantes son todos aquellos colectivos que van a jugar un papel en la solución digital con *Blockchain*. Estos incluyen desde las compañías que administran la red (en el caso de que las haya) hasta los usuarios de a pie, pasando por entidades auditoras, instituciones financieras, etc.

Para dimensionar el papel de cada participante, vale preguntarse: ¿Cuáles son los permisos que tendrá sobre la red? ¿Cómo va a interactuar con el sistema? ¿Tendrá acceso a una copia de toda la cadena? ¿Podrá ver solo las transacciones en las que participe o tendrá acceso a más información? ¿Cuáles son las transacciones que podrá realizar?

Dependiendo de cuáles sean las respuestas a estas preguntas, los participantes recibirán o no una copia de toda la cadena y tendrán o no permisos para ver y/o validar transacciones. Solo los participantes que tienen una copia de la cadena son considerados nodos. El resto, que accederán en general a través de un servicio web o una aplicación móvil, son simplemente usuarios.

2.1.1.2 Activos

Una vez tenemos claro quiénes van a ser los participantes, necesitamos saber qué van a intercambiar a través de la red *Blockchain*. La forma de entender este grupo es pensar que cuando los participantes hacen una transacción, en muchas ocasiones están transfiriendo algo. Ese “algo” es el activo, y puede ser un documento, un certificado, un informe, un token, una moneda digital, etc.

Es importante también resaltar que los documentos pesados no se almacenan en el *Blockchain* si no en bases de datos conectadas al *Blockchain*, lo que permite igualmente registrar sus modificaciones. Esto está motivado por cuestiones de eficiencia, ya que todos los nodos tienen una copia actualizada de la cadena y la ineficiencia es directamente proporcional al peso de ésta.

2.1.1.3 Transacciones

El tercer elemento son las transacciones. Si bien ya sabemos quién va a "jugar" (los participantes) y cuáles van a ser los "juguetes" (los activos), falta definir cuáles van a ser las reglas del juego. Las transacciones son la forma en la que se registra cualquier modificación en el *Blockchain*, desde el cambio de permisos de un usuario en el sistema, hasta la emisión de un certificado o el envío de una transferencia económica. Pueden verse también como las operaciones mediante las cuales los participantes crean, intercambian, modifican o destruyen activos.

A medida que se produce una transacción, se registra como un "bloque" de datos [9]. Estas transacciones muestran el movimiento de un activo, el cual puede ser tangible (un producto) o intangible (intelectual). El bloque de datos puede registrar la información de su elección: quién, qué, cuándo, dónde, cuánto e incluso la condición, como la temperatura de un envío de alimentos.

Cada bloque está conectado al bloque anterior y al bloque posterior. Estos bloques forman una cadena de datos a medida que un activo se mueve de un lugar a otro o cambia de dueño. Los bloques confirman tanto el tiempo exacto como la secuencia de las transacciones y se unen de forma segura para evitar que se alteren o se inserten entre dos bloques existentes.

Las transacciones se unen y forman una cadena irreversible: un *Blockchain*. Cada bloque adicional refuerza la verificación del bloque anterior y, por lo tanto, de todo el *Blockchain*. Esto hace que dicha cadena sea a prueba de manipulaciones, lo que constituye la ventaja principal de la inalterabilidad. Esto evita que alguien malintencionado modifique la cadena y crea un libro mayor distribuido de transacciones en el que se puede confiar.

La cadena de bloques es un registro de todas las transacciones, almacenadas y compartidas de forma pública. Los llamados mineros[4] se encargan de verificar esas transacciones. Tras ello, se incluye la nueva transacción en la cadena de bloques y se distribuye a los nodos que forman la red para que actualicen su cadena de bloques. Todo el mundo tiene una copia que se actualiza automáticamente; las alteraciones deben ser verificadas por todos en la red. Ninguna computadora controla los datos y cambiarlos en un bloque significaría que toda la cadena debe hacer lo mismo.

2.1.2 Beneficios más destacados del Blockchain

Blockchain representa una gran innovación en el registro y distribución de la información, eliminando así la necesidad de intermediarios. Por ello, esta tecnología tiene un gran potencial y traerá en un futuro no muy lejano grandes beneficios a todas las áreas de una organización [10]:

- **Seguridad**: la tecnología *Blockchain* permite realizar transacciones de forma segura y confiable, sin necesidad de intermediarios. Además, todos los miembros de la cadena deben llegar a un acuerdo acerca de los datos.
- **Gran potencial:** además de las transacciones financieras, esta tecnología se puede usar para muchas otras aplicaciones por su nivel de seguridad.
- **Mayor agilidad en las operaciones**: las transacciones se ejecutan a través de contratos inteligentes que se ejecutan por sí solos, de forma automática, digital y autónoma.
- **Datos inalterables:** las transacciones son inalterables, ya que se registran de forma permanente. Por ello, es una tecnología extremadamente complicada de hackear o modificar.
- **Resguardo de datos:** las empresas pueden crear *backups* de sus datos y asegurarse de tener la información bien guardada en varios bloques de la cadena *Blockchain*.

[4] Usuarios que utilizan su potencia informática para procesar transacciones y obtener recompensas, en este caso criptomonedas.

2.1.3 Tipos de redes de Blockchain

Existen al menos dos tipos de *Blockchain* principales; y estas dos generan un tercer tipo [11]; Ellas son:

- **Blockchain pública**
- **Blockchain privada**
- **Blockchain hibrida**

Cada una de ellas ofrece distintas posibilidades; mas no compiten entre sí. Las públicas operan abiertamente; es decir, cualquier individuo puede acceder a ellas siempre que tenga conexión a Internet. Las privadas, como su palabra lo indica; operan con una red cerrada a un grupo de individuos que han sido aceptados por un administrador central. Ambas tienen en común el uso de algoritmos de permisos o de consensos, a fin de poder validar cada una de las transacciones que se realizan. El último tipo de *Blockchain*, las híbridas son una mezcla entre las dos primeras ya que usan la tecnología de una *Blockchain* privada pero guardan el hash de los bloques en una *Blockchain* pública.
A continuación, se analizan las características, ventajas y desventajas de cada una de ellas.

2.1.3.1 Blockchain pública

Como se mencionó anteriormente, es una red de infraestructura pública; es decir, cualquier persona tiene la libertad de unirse a la red sin permiso alguno. Los participantes que se encuentran en la red pueden visualizar e incluso participar y validar las transacciones.

Las más relevantes actualmente son aquellas que están ligadas a las criptomonedas Ethereum y Bitcoin. Mayormente la cadena de bloques pública está enfocada para aquellas aplicaciones abiertas al público general, en las que se requiere una transparencia y descentralización total.

Habitualmente, se anima a los integrantes de la red a actuar como comprobadores de las transacciones llevadas a cabo, proporcionándoles a cambio alguna recompensa, por ejemplo, en forma de criptomoneda. Este tipo de individuos son denominados mineros y hacen uso de potentes GPUs (Unidad de Procesamiento Gráfico, del término anglosajón *Graphics Processing Unit*), que consumen gran cantidad de energía, para llevar a cabo estos procedimientos.

Ventajas

- **Transparencia**: cada participante tiene la libertad de ver u obtener copias de las transacciones, ya que se encuentran libremente distribuidas por la red.
- **Accesible**: Cualquier persona puede unirse a la red de nodos y acceder a todo el historial de transacciones e información.
- **Descentralizada y resistente a censuras**: por ser redes públicas, todos los participantes ejercen el mismo derecho. Nadie está a cargo o tiene algún poder en especial. Por lo tanto, tampoco pueden ser anuladas, cerradas o alteradas; ni las redes, ni las transacciones registradas.
- **Seguridad absoluta:** Nadie tiene la capacidad de cambiar o manipular ningún dato una vez éste ya ha sido registrado.

Desventajas

- **Alto consumo energético**: las redes públicas que usan el *Proof of Work* (PoW)[5] poseen esta gran desventaja. Implica valores energéticos muy altos y ambientalmente perjudiciales.
- **Trazabilidad en sus transacciones**: como son públicas, todas las transacciones pueden ser analizadas, estando o no asociadas a un usuario concreto (pueden estar anonimizadas).
- **Comisiones:** Los mineros son recompensados con una retribución a cambio de asegurar la red, con lo cual los usuarios que deseen registrar una transacción o información pagarán una comisión que irá a parar a estos mineros.

[5] Mecanismo de consenso (necesario la aceptación y verificación por todos los usuarios en la red) para garantizar que los bloques solo se consideren válidos si requieren una cierta cantidad de potencia computacional para producir.

2.1.3.2 Blockchain privada

Las cadenas de bloques que son privadas permiten o admiten participantes que han sido invitados a la red previamente. Lo que quiere decir que una entidad central asigna o permite qué participantes realizan algún tipo de transacción o labor extra de creación de bloques dentro de la misma.

Son redes convenientes para organizaciones, empresas o negocios que buscar tener un acceso limitado y privado a los registros; con el fin de mantenerlo fuera del alcance del público. Sin embargo, las operaciones o transacciones que se realizan son validadas por los mismos participantes.

Uno de los desarrollos de *Blockchain* privadas más importantes del mundo criptográfico es **Hyperledger**[6]. Este proyecto iniciado por la **Fundación Linux**[7] y varias empresas del sector tecnológico es el mayor ejemplo de *Blockchain* privada.

Ventajas

- **Un rendimiento mayor**: por ser privadas, son más pequeñas que las públicas; por ende, poseen un mayor rendimiento y velocidad en todas sus operaciones y un mayor alcance de disponibilidad en la misma red.
- **Confiable**: en las redes públicas los participantes se mantienen en anonimato. No obstante, en las privadas se identifica claramente a cada uno de ellos; por lo que representa un mayor nivel de confiabilidad.
- **No hay comisiones:** Los creadores de bloques no lo son por incentivos económicos sino por formar parte de la red, así pues, no son recompensados con una retribución en forma de criptomoneda, y por lo tanto, los usuarios no deben pagar comisión para usar la red.

Desventajas

- **Falta de descentralización**: al no estar descentralizada, los registros se encuentran totalmente con acceso cerrado. Así, es administrado por una sola organización o entidad según corresponda.
- **No es inmutable**: es decir, que el conjunto de nodos se puede poner de acuerdo para alterar las transacciones y los datos registrados en la cadena de bloques.

2.1.3.3 Blockchain híbrida

Las *Blockchain* híbridas surgen de la unión de una *Blockchain* privada y una pública. Más concretamente consiste en una *Blockchain* permisionada corriente pero con la diferencia que una referencia de cada bloque (hash) se guarda en una *Blockchain* pública. Cómo en el caso de las *Blockchain* privadas, las *Blockchain* híbridas, necesitan de un permiso para poder entrar o unirse a ella y ejecutar distintas funciones dentro de la misma.

Estas cadenas no son tan conocidas como las públicas y privadas. Son ideales para empresas, organizaciones o negocios que deseen tener una capa de seguridad adicional.

2.2 Internet of Things

El **Internet de las Cosas**, o *Internet of Things* por sus siglas en inglés, es una realidad y no una visión futurista. Podría definirse como la agrupación e interconexión de sistemas de *hardware* o dispositivos físicos a través de una red [12]. El IoT es capaz de interactuar sin intervención humana, compartir información y datos específicos entre ellos y con otros sistemas en tiempo real. Este concepto se conoce como comunicaciones de máquina a máquina (M2M, del término anglosajón *Machine to Machine*). A los llamados "objetos inteligentes" que constituyen las redes de sensores/actuadores inalámbricos (WSN, en inglés, *Wireless Sensor Networks*) se les otorga direcciones IP para que sean parte integral de Internet y puedan aprovechar los servicios que este ofrece en relación con la monitorización y control de dichos dispositivos.

---

[6] https://www.hyperledger.org/

[7] https://www.linuxfoundation.org/

El IoT introduce un cambio radical en la calidad de vida de las personas, ofreciendo nuevas oportunidades de acceso a datos, servicios específicos en la educación, en seguridad, asistencia sanitaria, el transporte o la industria (muchas veces se habla de la Industria 4.0), entre otros campos.

Como se introdujo previamente, en este trabajo se considera la habitual división en capas de IoT en torno a *edge*, *fog* y *cloud* [13]. Cada una de estas capas de detalla a continuación.

2.2.1 Edge

Como se ha introducido previamente, en el *edge* se encuentran los dispositivos que se encargan de recopilar la información del entorno, estando habitualmente limitados en capacidad de cómputo. Tradicionalmente, la información recopilada por el *edge* era enviada directamente a capas superiores, sin embargo, en los últimos años surgió la tendencia del *edge computing* [14], que se centra en procesamiento de dicha información en el propio *edge*, aunque sigue siendo posible su envío hacia capas superiores para procesamientos más profundo. Esta estrategia mejora la eficiencia evitando sobrecargar las infraestructuras y brinda una mejor experiencia al usuario porque se reduce la latencia creada cuando los datos tienen que viajar largas distancias. Además, mitiga los riesgos de seguridad y soberanía de los dato**s,** puesto que se reduce la cantidad de datos enviados.

Así, la implantación del *edge computing* requiere de encontrar un equilibrio entre lo que se procesa en el *edge* y lo que se procesa en capas superiores [15]. En resumen, una estrategia de tipo *edge computing* tiene sentido para:

- Tomar decisiones rápidas usando datos tan cerca de su fuente como sea posible y, así, evitar la latencia de la red.
- Optimizar flujos de datos a la nube procesando datos brutos en el *edge* y subiendo a la nube únicamente datos de alto valor.
- Tener acceso offline a analíticas de datos cuando el acceso a la nube o a la red no es fiable.

2.2.2 Fog

El *fog* [16] es una arquitectura prometedora que permite que los datos pueden ser analizados, procesados y almacenados por un conjunto de múltiples dispositivos informáticos de relativo bajo consumo, comúnmente denominados *fog nodes* (FN) o nodos de niebla. Estos nodos son económicos y permiten una baja latencia, ya que se encuentran ubicados más cerca de los distintos dispositivos de *edge* que del *cloud*. Nótese que algunos autores no consideran la existencia como tal del *fog*, identificando como *edge* a estos nodos intermedios, debido a que tecnológicamente son nodos muy similares a los encontrados en el *edge*. Sin embargo, los nodos del *fog* llevan a cabo tareas más propias de aquellas realizadas en el *cloud*, que, en el *edge*, como por ejemplo el almacenamiento de información o el procesamiento profundo de la información. Por ello, el *fog* se podría definir como una capa *cloud* limitada o reducida. Así, en este trabajo, consideraremos que aquellos dispositivos intermedios entre el *edge* (captura de datos) y el *cloud* pertenecen a la capa *fog*.

2.2.3 Cloud

El *cloud computing* o la **computación en la nube** consiste en el suministro de recursos informáticos a petición, desde aplicaciones hasta centros de datos, a través de Internet y con un modelo de pago según uso [17].

La computación en la nube ofrece a los individuos y a las empresas de todos los tamaños, en cualquier dispositivo, lugar y momento, la capacidad de un pool de recursos de computación con buen mantenimiento, seguro, de fácil acceso y bajo demanda. Presenta una solución ideal para el manejo de los flujos enormes de datos y su procesado para un número sin precedentes de dispositivos IoT y solución de aplicaciones.

Existen tres principales categorías de servicios proporcionadas por los proveedores de la nube [18] [19]:

- Infraestructura como servicio (**IaaS,** del término anglosajón *Infrastructure as a Service*): El usuario alquila, paga en función del uso, la infraestructura. Accede a ésta con una API o un panel. El usuario

gestiona el sistema operativo, las aplicaciones, datos y el *middleware*[8]. Los proveedores de servicios de la nube se encargan de los sistemas de *hardware*, las redes, los discos duros, el almacenamiento de datos y los servidores. Son los responsables de prevenir las interrupciones, hacer reparaciones y solucionar los problemas de *hardware*. Las máquinas virtuales de Azure, como la que se ha utilizado en este proyecto, son infraestructura como servicio.

- Plataforma como servicio (**PaaS,** del término anglosajón *Platform as a Service*): es un servicio que ofrece a los desarrolladores una plataforma *cloud* completa (*hardware*, *software* e infraestructura) para que los clientes solo se tengan que preocupar en crear aplicaciones gestionar los datos en los que se basan. Eliminando la complejidad y la rigidez que tendría la compra y mantenimiento de componentes en una plataforma local. Ejemplos: Microsoft Azure, Amazon Web Service (AWS) y Google Cloud.
- Software como servicio (**SaaS,** del término anglosajón *Software as a Service*): es un servicio que ofrece una aplicación de *software* gestionada, mantenida, actualizada y desarrollada por el proveedor de servicios de nube. Las aplicaciones SaaS son aplicaciones web o móviles a las que los usuarios pueden acceder a través de un navegador web y no requieren instalar ninguna aplicación en su ordenador. Se conoce también como *software* bajo demanda. Los ejemplos comunes son el correo electrónico Outlook y Microsoft Office 365.

2.3 Cifrado

En esta sección se pretende contextualizar e introducir algunos conceptos asociados a la seguridad. El cifrado simétrico, concretamente el algoritmo AES, se emplea para cifrar los datos generados en ambas soluciones desarrolladas para que luego sean guardados de forma segura en los bloques de *Blockchain*. Aunque como se verá a continuación, este tipo de cifrado tiene el inconveniente de que no existe una forma segura de compartir la clave de encriptación, sin embargo, tiene un gran rendimiento en el cifrado y descifrado de grandes volúmenes de datos.

Cuando se comparte datos confidenciales o personales por canales de comunicación no seguros, como puede ser Internet, es posible que la información pueda ser comprometida por los intrusos pasivos o activos. La criptografía es la ciencia que hace uso de métodos y herramientas matemáticas con el objetivo principal de cifrar[9] los datos. La palabra criptografía proviene del griego y etimológicamente significa «escritura secreta» [20].

La encriptación o cifrado es un mecanismo de seguridad que permite modificar o codificar un mensaje en texto plano a texto cifrado (cadena de letras, números y símbolos) para que la información sea un sin sentido y solamente las personas autorizadas puedan entenderlo. El cifrado implica utilizar una clave criptográfica; un conjunto de valores matemáticos que acuerdan tanto el emisor como el receptor. El destinatario utiliza la clave para descifrar el mensaje y transformarlo a su formato original [21].

La seguridad de la encriptación reside en la clave y su longitud, pues según aumenta ésta última, crecen los posibles valores que pueda tomar y, por lo tanto, también el esfuerzo que supone romper dicho cifrado.

2.3.1 Cifrado simétrico

La criptografía simétrica o criptografía de clave secreta se caracteriza por emplear la misma clave para el cifrado y el descifrado de los mensajes entre en el emisor y el receptor. Es la técnica criptográfica más antigua que existe, pero sigue ofreciendo un alto nivel de seguridad y eficiencia cuando cifra grandes cantidades de datos sin tener un efecto negativo en el rendimiento [22].

En contra parte, presenta un problema difícil de resolver: la necesidad de compartir la clave entre las partes implicadas antes de establecer cualquier comunicación segura. Si un hacker intercepta la clave en el trayecto, podrá descifrar todos los mensajes cifrados [23]. Como limitación, habría que tener una clave distinta por

[8] Software que brinda servicios y funciones comunes a las aplicaciones, además de lo que ofrece el sistema operativo.

[9] Transcribir en guarismos, letras o símbolos, de acuerdo con una clave, un mensaje o texto cuyo contenido se quiere proteger.

cada individuo o entidad con la que se quisieran intercambiar mensajes cifrados.

2.3.2 Cifrado de asimétrico

En la criptografía simétrica era necesario compartir las claves y no había una forma segura de hacerlo. En la década de 1970, nace el cifrado asimétrico o PKI Infraestructura de clave pública, en donde cada usuario de la comunicación tiene un par de claves relacionadas matemáticamente: una clave pública que será accesible por todos y una clave privada que debe ser protegida por su propietario [24].

Un mensaje cifrado con la clave pública de un destinatario no puede ser descifrado por nadie, excepto por el poseedor de la clave privada correspondiente, presumiblemente su propietario. La clave privada genera firmas digitales y con la clave pública se puede verificar la autenticidad de esta. Con el cifrado asimétrico se suprime la necesidad del envío de la clave, pero se ralentiza el proceso de cifrado [25].

2.3.3 Algoritmo AES

El algoritmo AES (*Advanced Encryption Standard*), conocido como Rijndael, es un estándar de cifrado simétrico por bloque, ya que divide la información a cifrar en secciones llamadas bloques. Creado en Bélgica y adoptado por el gobierno de los Estados Unidos para remplazar el algoritmo DES (*Data Encryption Standard*) de 56 bit [26].

Con un tamaño de clave de 128, 192 o 256 bits y un tamaño de bloque de 128 bits, se puede afirmar que el algoritmo AES es seguro, rápido y eficiente. Por eso es el algoritmo de cifrado por bloques más utilizado de la actualidad y la mayoría de los procesadores incluyen instrucciones para cifrar y descifrar información con este método. El nivel de seguridad que ofrece el algoritmo AES es tan alto que la Agencia de Seguridad Nacional de Estados Unidos (NSA) lo utiliza para encriptar sus documentos clasificados como «*Top Secret*».

El algoritmo se basa en varias sustituciones, permutaciones y transformaciones lineales, cada una ejecutada en bloques de datos de 16 bytes. Esas operaciones, llamadas rondas, se repiten 10, 12 o 14 veces dependiendo del tamaño de la clave de cifrado. Durante cada ronda, una clave circular única se calcula a partir de la clave de cifrado y se incorpora en los cálculos. El cambio de un solo bit ya sea en la clave, o en el bloque de texto sin cifrado, da como resultado un bloque de texto cifrado completamente diferente [27].

2.4 Bases de Datos

Una **base de datos** es una colección estructurada de datos a los que se puede acceder de forma digital. Trabaja en conjunto con un *software* denominado **sistema gestor de base de datos *(DBMS***, del término anglosajón *Data Base Management System)*, el cual permite ordenar, modificar y consultar la información contenida en un banco de datos [28].

Características principales de una base de datos:

- **Los datos están ordenados:** La información contenida en una base de datos es estructurada y organizada de acuerdo con ciertos criterios que depende generalmente del tipo de base de datos.
- **Trabaja junto a un gestor de base de datos:** Para administrar una base de datos, se usa lo que se conoce como sistema de gestión de base de datos, una herramienta que permite almacenar, estructurar, modificar, acceder y consultar información.
- **Permite almacenar grandes volúmenes de datos:** Un banco de datos tiene la capacidad de guardar una gran cantidad de información.
- **La información contenida puede consultarse rápidamente:** Los datos almacenados pueden ser consultados con gran rapidez, independientemente de la dificultad de la consulta. Y aunque pueda haber bases de datos con lentitud en sus consultas, estos pueden ser optimizados para mejorar sus tiempos de respuestas.
- **Es seguro, si se toman las medidas adecuadas:** Una base de datos puede ser un sistema seguro siempre y cuando se apliquen las técnicas correctas para garantizar su solidez.

En términos generales, una base de datos presenta los siguientes componentes:

- **Tablas o estructuras de datos:** Para guardar datos.
- **Consultas:** Para acceder a la información contenida en una base de datos. Con las consultas se pueden modificar, agregar, eliminar y mostrar datos.
- **Informes:** Para presentar información contenida en una base de datos. Esta información se muestra de una forma en concreto y en un determinado formato para que pueda ser imprimida.
- **Formularios:** Para ingresar datos.
- **Macros:** Para crear instrucciones y ejecutar funciones.

A continuación, se analizan los dos tipos de bases de datos más habituales en el ámbito, las bases de datos relacionales y las no relacionales.

2.4.1 Modelo relacional

En los primeros años de las bases de datos, cada aplicación almacenaba datos en su propia estructura única. Cuando los desarrolladores querían crear aplicaciones para usar esos datos, tenían que conocer muy bien esa estructura de datos concreta a fin de encontrar los datos que necesitaban. Esas estructuras de datos eran poco eficaces, el mantenimiento era complicado y era difícil optimizarlas para ofrecer un buen rendimiento en las aplicaciones [29].

El modelo de base de datos relacional se diseñó para resolver el problema causado por estructuras de datos múltiples y arbitrarias. Proporcionó una forma estándar de representar y consultar datos que podría utilizar cualquier aplicación. Desde el principio, los desarrolladores se dieron cuenta de que la virtud principal del modelo de base de datos relacional era el uso de tablas, ya que era una forma intuitiva, eficiente y flexible de almacenar y acceder a información estructurada.

Una base de datos relacional es un tipo de base de datos que almacena y proporciona acceso a puntos de datos relacionados entre sí. Las bases de datos relacionales se basan en el modelo relacional, una forma intuitiva y directa de representar datos en tablas. En una base de datos relacional, cada fila en una tabla es un registro con una ID única, llamada clave. Las columnas de la tabla contienen los atributos de los datos y cada registro suele tener un valor para cada atributo, lo que simplifica la creación de relaciones entre los puntos de datos.

Las bases de datos relacionales tienen algunas características que las distinguen de otros tipos de bases de datos. Son las siguientes:

- Una base de datos relacional consta de una o más tablas de dos dimensiones de valores de datos, y la construcción de dichas tablas viene definida por reglas muy simples. Cada tabla corresponde a un tipo de entidad conceptual, que no contenga grupos de repetición.
- Las relaciones entre dos o más tablas se establecen en virtud de los valores de datos comunes contenidos en las tablas correspondientes.
- Se ha desarrollado una metodología sistemática para la transformación de un conjunto de tipos de entidad (que representan un diseño conceptual) en un conjunto correspondiente de tablas. El objetivo de esta metodología es la generación de un grupo de tablas en el cual se minimice la duplicación de datos, y en el que se hayan eliminado determinados problemas asociados con el mantenimiento de las tablas.
- Los sistemas de bases de datos relacionales suelen tener asociados lenguajes de consulta de alto nivel, como por ejemplo el habitual SQL (*Structured Query Languaje*)[10], que facilitan la realización de búsquedas y de actualizaciones en la base de datos, de la manera más flexible posible.

Los dos primeros puntos constituyen el núcleo del modelo relacional, ya que definen las características fundamentales que distinguen este modelo de los demás. Aunque los dos puntos últimos suelen considerarse como parte del modelo relacional, no son exclusivos de éste, y, de hecho, a menudo pueden ser usados muy

[10] https://es.wikipedia.org/wiki/SQL

ventajosamente en otros tipos de bases de datos.

2.4.2 Modelo no relacional

Las bases de datos no relacionales son un sistema de almacenamiento de información que se caracteriza por no usar el lenguaje SQL para las consultas, así como almacenar la información en forma de colecciones y documentos. Esto no significa que no puedan usar el lenguaje SQL, pero no lo hacen como herramienta de consulta, sino como apoyo. Por ello también se les suele llamar NoSQL o «no solo SQL» [30].

Las bases de datos no relacionales son más actuales que las relacionales, y su desarrollo se ha basado en la necesidad de crear sistemas de gestión capaces de trabajar con datos no estructurados o semi-estructurados.

Las principales características de una base de datos no relacional son las siguientes:

- La información no se almacena en tablas sino a través de documentos.
- Son bases de datos muy útiles para organizar y gestionar información no estructurada, o cuando no se tiene una noción clara de los datos a almacenar.
- Son bases de datos con alto grado de escalabilidad y están diseñadas para soportar grandes volúmenes de datos.
- No utilizan el lenguaje SQL para consultas, aunque sí lo pueden usar como herramienta de apoyo.
- Es un sistema de almacenamiento de datos relativamente nuevo, y como tal, todavía no posee un sistema estandarizado.
- A diferencia de las no relacionales, no garantizan el cumplimiento de las cualidades ACID, esto es, atomicidad, consistencia, integridad y durabilidad.

3 Arquitecturas propuestas

En este capítulo, se realiza una descripción a alto nivel de cada uno de los componentes que conforman el sistema y se describirá la estructura del modelo de datos diseñado. La Figura 3-1 muestra un diagrama donde se encuentran los componentes de ambas soluciones desarrolladas (solución 1 y 2, en violeta y naranja, respectivamente).

La solución 1 consta de una Raspberry Pi y la máquina virtual de Azure. En la Raspberry Pi, dispositivo del *fog*, se ejecuta un programa en Python que genera la información (cadenas de texto aleatorias) que serán enviadas al *cloud*, la máquina virtual de Microsoft Azure, de forma asegurada mediante *Blockchain*. La comunicación entre estos dispositivos se realiza por SSH. Las cadenas y los datos estadísticos tomados en el proceso son insertados en una base de datos en *MongoDB* que se encuentra almacena en la MV. En la solución 2, se incluye un dispositivo del *edge*, un nodo ESP32, que se encargará de generar la información aleatoria (el lugar de generarla en la Raspberry). La comunicación entre el ESP32 y la Raspberry se llevará a cabo mediante WiFi. La arquitectura de esta solución 2 es técnicamente similar a la seguida en Bindi.

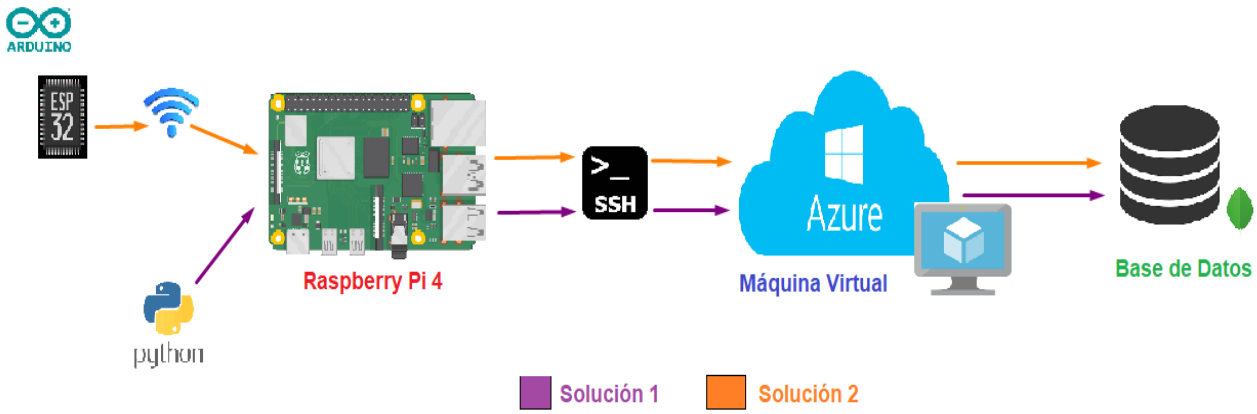

Figura 3-1 Arquitectura del sistema

3.1 Componentes del sistema

En esta sección se describen, a nivel técnico, los principales componentes *hardware* y *software* que conforma las soluciones desarrolladas. Estos son, el nodo del *edge* ESP32, la placa de desarrollo Raspberry Pi4, la máquina virtual Azure y el sistema gestor de bases de datos no relacional *MongoDB*.

3.1.1 ESP32

El ESP32 pertenece a la familia de chips SoC (por sus siglas en inglés, *System on a Chip*). Fue desarrollado por la empresa china ESPRESSIF System[11]. El chip integrado, está diseñado para ser escalable y adaptable. Esta placa permite controlar de forma eficiente y económica todo tipo de sensores, módulos y actuadores mediante WiFi, Bluetooth y Bluetooth Low Energy (BLE) [12], en proyectos de IoT. Fue sucesor del famoso ESP8266. Agrega funcionalidad y versatilidad invaluables a las aplicaciones con requisitos mínimos de placa de circuito impreso. Presenta un diseño robusto y está alimentado por circuitos de calibración avanzados que pueden eliminar dinámicamente las imperfecciones del circuito externo y adaptarse a los cambios en las condiciones externas. Además, es compatible con el cifrado/descifrado de *hardware* basado en AES para proteger los programas y datos de los desarrolladores en flash.

[11] https://www.espressif.com/

[12] diseñado para mantener un rango de alcance de comunicación similar y un bajo consumo de energía con respecto al Bluetooth clásico.

El puerto micro USB de tipo B permite su programación y/o suministrar la energía. Esta pequeña placa, que dispone de 38 pines, está diseñada especialmente para trabajar con un *protoboard* o placa de prototipado y con ayuda de los cables Dupont se puede conectar rápidamente con todo tipo de sensores. A continuación, la Figura 3-2 muestra una imagen con la distribución de pines o **pinout** de esta placa de desarrollo.

Figura 3-2 Pines del ESP32 [31]

El ESP32 puede ser programado con gran variedad de *softwares*, lenguajes de programación, *frameworks*, librerías, código/ejemplos etc. Los más comunes a elegir son: Esp-idf (Espressif IoT Development Framework)[13], Arduino IDE[14] (en lenguaje C++), y Simba Embedded Programming Platform[15] (en lenguaje C), MicroPython[16]. En este proyecto se ha optado por utilizar el Arduino IDE para su implementación.

Las características del ESP32 incluyen [32]:

- **Procesador**:
  - CPU: microprocesador de 32-bit Xtensa LX6 de doble núcleo (o de un solo núcleo), operando con una frecuencia de reloj de 80 a 240 MHz
  - Co-procesador de ultra baja energía (ULP)
- **CPU y memoria interna**:
  - 520 KByte SRAM
  - 448 KB de ROM para funciones básicas y de arranque.
  - 8 KB de SRAM en RTC, que se denomina memoria RTC FAST y se puede utilizar para el almacenamiento de datos; la CPU principal accede a él durante el arranque RTC desde el modo de suspensión profunda.
  - 8 KB de SRAM en RTC, que se denomina memoria RTC SLOW y el coprocesador puede acceder a ella durante el modo de suspensión profunda.
  - 1 Kbit de eFuse: 256 bits se utilizan para el sistema (dirección MAC y configuración del chip) y los 768 bits restantes se reservan para las aplicaciones del cliente, incluido el cifrado flash y la identificación del chip.
- **Conectividad inalámbrica**:
  - Wi-Fi: 802.11 b/g/n (802.11n hasta 150 Mbit/s). Rango de frecuencia @ 2.4 GHz

[13] https://www.espressif.com/en/products/sdks/esp-idf
[14] https://www.arduino.cc/en/software
[15] https://simba-os.readthedocs.io/en/latest/
[16] https://micropython.org/

  - Bluetooth: v4.2 BR/EDR y BLE
- **Interfaces periféricas**:
  - SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC, Two-Wire Automotive Interface (TWAI®), compatible with ISO11898-1 (CAN Specification 2.0)
- **Hardware**:
  - On-chip sensor: Hall sensor
  - Integrated crystal: 40 MHz crystal
  - Integrated SPI flash: 4 MB[17]
  - Operating voltage/Power supply: 3.0 V ~ 3.6 V
  - Operating current Average: 80 mA
  - Minimum current delivered by power supply: 500 mA
  - Recommended operating ambient temperature range: –40 °C ~ +85 °C

Además del ESP32 se valoró el uso de otras plataformas, como el Arduino Uno Rev3 [18]. Los motivos que llevaron a la selección del ESP32 se deben a que cumplía con todos los requisitos *hardware* para la solución del *edge* en este trabajo, tenía un mejor precio y el candidato había trabajo con esta plataforma en la asignatura de Computación Ubicua. Además, el ESP32 tiene integrado el módulo WiFi y Bluetooth, por lo que no era necesario conectar otros componentes externos para lograr esta funcionalidad.

3.1.2 Raspberry Pi

La **Raspberry Pi** es una serie de placa simple (SBC) de bajo costo desarrollado por la fundación Raspberry Pi de un grupo de académicos de Cambridge**.** Surge con el objetivo de estimular la enseñanza de la informática en las escuelas y poner en manos de las personas de todo el mundo el poder de la computación y la creación digital.

No se indica expresamente si es *hardware* libre o con derechos de marca, pero cualquiera puede convertirse en revendedor o redistribuidor de las tarjetas permitiendo su uso libre tanto a nivel educativo como particular. El *software* es de código abierto, siendo *Raspberry Pi OS* (anteriormente Raspbian) su sistema operativo oficial, basado en una distribución GNU/Linux, concretamente en Debian. La herramienta *Raspberry Pi Imager* permite instalar en una tarjeta microSD este sistema operativo.

La popularidad y los proyectos no han dejado de aumentar desde que llegaron al mercado, allá por el 2012. Alguno de los proyectos que se pueden hacer son los siguientes: servidor de archivos, streaming o web. Media Center. Cliente Torrent. Control de dispositivos Arduino. Domótica para automatizar la casa y un sistema de videovigilancia etc.

Raspberry Pi 4 Model B es el último producto de la popular gama de computadoras Raspberry Pi. Ofrece un aumento sin precedentes en la velocidad del procesador, el rendimiento multimedia, la memoria y la conectividad en comparación con la generación anterior de Raspberry Pi 3 Modelo B+, manteniendo la compatibilidad con versiones anteriores y un consumo de energía similar. Para el usuario final, el Raspberry Pi 4 Model B ofrece un rendimiento de escritorio comparable a un computador en tareas ofimáticas sencillas.

[17] se conecta a GPIO6, GPIO7, GPIO8, GPIO9, GPIO10 y GPIO11. Estos seis pines no se pueden usar como GPIO normales.
[18] https://docs.arduino.cc/hardware/uno-rev3

Los componentes de la Raspberry Pi 4B se pueden observar en la Figura 3-3. Sus principales características son como siguen [33]**:**

- **Procesador**:
  - SoC Broadcom BCM2711, Cortex-A72 de cuatro núcleos (ARM v8) de 64 bits a 1,5 GHz.
- **Memoria y Hardware**:
  - 4 GB LPDDR4 con ECC integrado.
  - GPIO: Encabezado GPIO de 40 pines estándar de Raspberry Pi (totalmente compatible con versiones anteriores de placas anteriores).
  - Soporte de tarjeta SD: Ranura para tarjeta micro-SD para cargar sistema operativo y almacenamiento de datos.
  - Recomendado operar en un rango de temperatura de 0 – 50 grados ºC.
- **Conectividad**:
  - LAN inalámbrica de 2,4 GHz y 5,0 GHz IEEE 802.11ac inalámbrico.
  - Bluetooth 5.0, BLE.
  - Gigabit Ethernet.
  - 2 puertos USB 3.0.
  - 2 puertos USB 2.0.
- **Sonido del video**:
  - 2 × puertos micro-HDMI (compatible con hasta 4kp60).
  - Puerto de pantalla MIPI DSI de 2 carriles.
  - Puerto de cámara MIPI CSI de 2 carriles.
  - Puerto de audio estéreo y video compuesto de 4 polos.
- **Multimedia**:
  - H.265 (descodificación 4kp60).
  - H264 (descodificación 1080p60, codificación 1080p30).
  - OpenGL ES, gráfico 3.0.
- **Potencia de entrada**:
  - 5V DC mediante conector USB-C (mínimo 3A[19]).
  - 5V DC a través del cabezal GPIO (mínimo 3A).
  - Alimentación a través de Ethernet (PoE) habilitada (requiere PoE HAT separado).

La Raspberry Pi 4 B fue seleccionada como arquitectura porque ofrece un rendimiento muy bueno a un buen precio. Posee una memoria RAM suficiente (4 GB), posibilidad de incorporar una tarjeta microSD con capacidad de almacenamiento deseado y una conectividad (WiFi) necesarias para ejecutar los programas que se detallaran en el próximo capítulo.

[19] Se puede utilizar una fuente de alimentación de 2,5 A de buena calidad si los periféricos USB descendentes consumen menos de 500 mA en total.

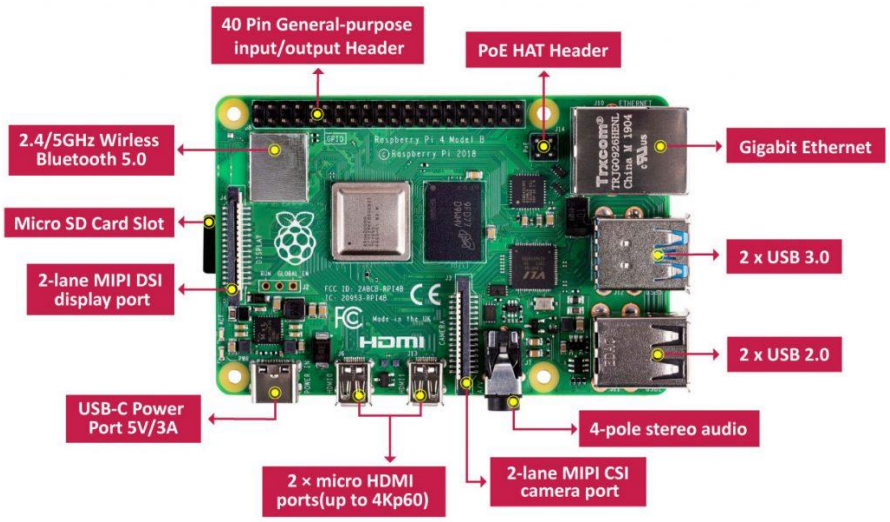

Figura 3-3 Raspberry Pi 4 Model B [34]

3.1.3 Máquina Virtual en Azure

Microsoft Azure[20] es una **nube pública de pago por uso** que te permite compilar, implementar y administrar rápidamente aplicaciones en una red global de *datacenters* (centros de datos) de Microsoft. Los centros de datos, disponibles en más de 60 regiones [21], pueden alojar las infraestructuras que se necesite: servidores físicos, redes, máquinas virtuales, plataformas de desarrollo, almacenamiento…

Microsoft Azure cuenta con más de 200 productos y servicios en expansión constante que ayudan a las empresas de mayor tamaño como a las pequeñas y medianas empresas a cumplir con los actuales y futuros desafíos. Compatible con muchos lenguajes, herramientas y marcos de programación diferentes, incluidos *software* y sistemas específicos de Microsoft y de terceros. La mayoría los servicios son escalables y son capaces de responder a necesidades generales y particulares proporcionando *software* como servicio (SaaS), plataforma como servicio (PaaS) e infraestructura como servicio (IaaS). Estos servicios están garantizados con una disponibilidad del 99.99%. Cuenta con todas las certificaciones en materia **seguridad y protección de datos**.

Azure Portal es una consola unificada basada en web que proporciona una alternativa a las herramientas de línea de comandos. Con Azure Portal, se pueden administrar todos los servicios y aplicaciones de la suscripción de Azure. Diseñado para proporcionar flexibilidad, resistencia (a los errores de centros de datos individuales y evita que se ralentice la red) y disponibilidad (no deja de actualizarse y no requiere tiempo de inactividad para las actividades de mantenimiento).

Una máquina virtual es un ordenador virtual (construido en *software*) en el que se puede instalar un **sistema operativo** y ejecutar aplicaciones. Proporciona la misma funcionalidad que los ordenadores físicos. Tiene su propia CPU, disco duro, RAM, tarjeta gráfica etc. Estas prestaciones *hardware* se "toman prestadas" del equipo físico o servidor remoto. Tiene la ventaja de permanecer completamente independiente al host físico.
Las máquinas virtuales en Azure ofrecen muchas ventajas, como las siguientes [22]:

- **Ahorro de costos**: aumenta los beneficios al ejecutar varios entornos virtuales en una única infraestructura, ya que reduce los costos en infraestructura, mantenimiento y electricidad.
- **Agilidad y velocidad**: la puesta en marcha de una máquina virtual es relativamente fácil y rápida.
- **Tiempo de inactividad reducido**: las máquinas virtuales son muy portables y fáciles de migrar de un hipervisor[21] a otro en un equipo diferente en caso de fallo.
- **Escalabilidad**: para aumentar la disponibilidad y el rendimiento de las aplicaciones, las máquinas

[20] https://azure.microsoft.com/es-es/

[21] Permite ejecutar diferentes sistemas operativos en varias máquinas virtuales al mismo tiempo

virtuales se pueden escalar más fácilmente en más servidores virtuales o físicos para distribuir la carga de trabajo.

- **Seguridad**: el uso de un sistema operativo invitado en una máquina virtual permite ejecutar aplicaciones de una seguridad dudosa y proteger el sistema operativo host.

Para este proyecto, se ha creado una máquina virtual en Azure con los siguientes detalles:

- Nombre del equipo: Linux-VM
- Sistema Operativo: Linux
- Servicio: Ubuntu Server
- Plan: 20_04-lts-gen2
- Publicador: canonical
- Tamaño: Standard D2s v3 (vCPU: 2, RAM: 8 GB)
- Grupo de recursos: cubanFlow-group
- Ubicación: France Central (Zona 3)
- Dirección IP pública: 20.199.10.61
- Dirección IP privada: 10.0.0.4
- Tipo de seguridad: Estándar

3.1.4 MongoDB

MongoDB[22] es un DBMS orientado a documentos que se utiliza para el almacenamiento de datos de gran volumen. Este gestor salió a la luz a mediados de la década de 2000 bajo la categoría NoSQL. Este tipo de DBMS utiliza esquemas dinámicos, lo que significa que puede crear registros sin definir primero la estructura, como los campos o los tipos y sus valores [35], lo que permite una alta flexibilidad.

Los datos guardados con *MongoDB* son de fácil lectura para los usuarios, almacenándolos en documentos bajo el formato JSON, el lugar de usar tablas como en el enfoque relacional. Los documentos permiten almacenar toda la información que se quiera, lo cual a veces puede ocasionar problemas en la consistencia de los datos.

MongoDB utiliza un lenguaje de consultas no estructurado, realizando las consultas especificando el nombre del documento con las propiedades que queremos filtrar. *MongoDB* se inclina a CP (Consistencia y Tolerancia) a particiones**,** esto significa que todos los clientes acceden a una vista consistente de la base de datos. Lo cual implica que los usuarios de un nodo deben esperar a que los otros nodos se sincronicen para poder ser visibles y editables, en este caso la disponibilidad queda en segundo plano frente a la consistencia.

La base de datos *MongoDB* contiene un conjunto de colecciones. Una colección no tiene un esquema predefinido y rígido como las tablas en el modelo relacional, y almacena los datos en documentos BSON o "Binary JSON"[23]. Un documento es un conjunto de campos y se puede considerar como una fila en una colección. Puede contener estructuras complejas, como listas, o incluso un documento completo. Cada documento tiene un campo de ID, que se usa como clave principal y cada colección puede contener cualquier tipo de documento.

En el ámbito de la seguridad, *MongoDB* utiliza un control de acceso basado en roles con privilegios flexibles**,** sus características de seguridad incluyen autenticación, auditoría y autorización. También permite el uso de TLS/SSL con el propósito de encriptar los datos y que solo sean accesibles para el cliente.

---

[22] https://www.mongodb.com/

[23] Formato empleado para el almacenamiento y transferencia de los datos en las bases de datos *MongoDB*. La estructura binaria de BSON codifica información de tipo y longitud lo que permite analizarla mucho más rápidamente.
https://www.mongodb.com/json-and-bson

A continuación, se presenta un análisis de ventajas y desventajas de *MongoDB*. Este análisis se completa con la tabla comparativa presentada en la Tabla 1, procedente de [35], donde se compara *MongoDB* y una conocida alternativa relacional.

Ventajas

- Validación de documentos
- Motores de almacenamiento integrado
- Menor tiempo de recuperación ante fallas

Desventajas

- No es una solución adecuada para aplicaciones con transacciones complejas
- No tiene un reemplazo para las soluciones de herencia
- Aún es una tecnología joven

MongoDB	MYSQL
Representa los datos como documentos JSON.	Representa los datos en tablas y filas.
No necesita definir el esquema. En su lugar, simplemente coloca documentos que ni siquiera necesitan tener los mismos campos.	Requiere que defina sus tablas y columnas antes de poder almacenar cualquier cosa, y cada fila de una tabla debe tener las mismas columnas.
Tiene una estructura predefinida que se puede definir y adherir, pero también, si necesita diferentes documentos en una colección, puede tener diferentes estructuras.	Utiliza el lenguaje de consulta estructurado (SQL) para acceder a la base de datos. No se puede cambiar el esquema.
Los lenguajes soportados son C++, C	Los lenguajes soportados son C++, C y JavaScript
El desarrollo continuo lo realiza *MongoDB*, Inc.	Oracle Corporation realiza un desarrollo constante.
Admite la replicación, la fragmentación y las elecciones automáticas integradas.	Admite la replicación maestro-esclavo y la replicación maestra.
Si no se encuentra un índice, se deben escanear todos los documentos dentro de una colección para seleccionar los documentos que ofrecen una coincidencia con la declaración de consulta.	Si no se define un índice, entonces el motor de la base de datos necesita escanear la tabla completa para encontrar todas las filas relevantes.
GPL v2/ Licencia comercial disponible OD	GNU AGPL v3.0/ Licencias comerciales disponibles OD
Si la mayoría de sus servicios están basados en la nube, es el más adecuado.	Si la seguridad de los datos es su prioridad, es la mejor opción.
No impone restricciones en el diseño del esquema.	Requiere que defina sus tablas y columnas antes de poder almacenar cualquier cosa. Cada fila de una tabla debe tener las mismas columnas.
Utiliza JavaScript como lenguaje de consulta.	Utiliza el lenguaje de consulta estructurado (SQL).
No es compatible con JOIN.	Admite operaciones JOIN.
Tiene la capacidad de manejar grandes datos no estructurados.	Bastante lento en comparación con *MongoDB* cuando se trata de grandes bases de datos.
Análisis en tiempo real, gestión de contenido, Internet de las Cosas, aplicaciones móviles	Datos estructurados con un esquema claro
No se requiere definición de esquema, por lo que el riesgo de ataque es menor debido al diseño	Riesgo de ataques de inyección SQL
Una opción ideal si tiene datos estructurados y/o no estructurados con potencial de rápido crecimiento.	Una excelente opción si tiene datos estructurados y necesita una base de datos relacional tradicional.

Tabla 1 Comparativa del MongoDB y MySQL [36]

Conectar dispositivos a Internet, recopilar y analizar sus datos, son la base fundamental del IoT. El crecimiento de esta tecnología en un mundo cada vez más conectado obliga a utilizar una base de datos optimizada y que permita almacenar grandes cantidades de información procedentes de un mundo altamente cambiante. Por lo que la flexibilidad es esencial. Bajo esa motivación, se optó por utilizar un modelo no relacional. Específicamente, se optó por *MongoDB* por ser escalable, de código abierto y proporcionar un alto rendimiento a las necesidades de IoT. Permitiendo manejar grandes volúmenes de datos, escalar horizontal o vertical y admitir de forma nativa datos de series temporales típicos en arquitecturas IoT

3.2 Estructura del modelo de datos

En este apartado se describe la estructura lógica de la base de datos del sistema y cómo se accede a ellos. La base de datos, la misma en ambas soluciones, se nombró como *db_TFG* y tiene una única colección llamada *Blockchain*. Los documentos de la colección *Blockchain*, siguiendo un formato JSON, contienen bloques (un bloque es cada uno de los mensajes que son enviados al *cloud*, ya sea desde el *edge* o el *fog*) que siguen la siguiente estructura:

```
Bloque : {
        _id : ObjectId,
        index : int32,
        time_stamp : String,
        msg : Array,
        hash : String,
        prev_hash : String,
        statistic : Object {
                ram_rasp : String,
                time_rasp : String,
                size_msg : String,
                size_msg_cipher : String,
                ram_mv : String,
                time_mv : String
        }
}
```

- **_id:** Tipo *ObjectId*. Identifica de forma única al bloque. Este dato es autogenerado por el sistema al insertar un nuevo bloque. El tipo *ObjectId* en *MongoDB* consiste en una representación compuesta por 12 bytes, donde 4 bytes lo conforma un valor timestamp en relación con el momento de creación del documento, 5 bytes un número aleatorio y 3 bytes para un contador autoincremental que parte de un valor inicial aleatorio.
- **index:** Tipo int32. Número cardinal que establece la posición en la que se ha guardado el bloque dentro de la cadena de bloques.
- **time_stamp:** Tipo String. Fecha y hora de creación del bloque.
- **msg:** Tipo Array. Almacena el contenido del mensaje encriptado enviado desde el *edge*/*fog*. Este mensaje estará formado por una o más cadenas aleatorias de caracteres.
- **hash:** Tipo String. comprobador del bloque. Este identificador único es calculado al realizar la suma del index, time_stamp, prev_hash y el contenido del msg. Este campo permitirá comprobar la validez del bloque.
- **prev_hash:** Tipo String. Campo hash del bloque anterior, con respecto al actual, introducido en la cadena de bloques.
- **statistic:** Tipo Object. Almacenará los campos estadísticos que se detallan a continuación. Estas variables son tomadas para analizar el rendimiento de la solución.
  - **ram_rasp:** Tipo String. Valor de la RAM consumida en la Raspberry Pi para encriptar los mensajes mediante AES de los bloques.
  - **time_rasp:** Tipo String. Tiempo empleado para encriptar los mensajes mediante AES de los bloques en la Raspberry Pi.
  - **size_msg:** Tipo String. Tamaño total de los mensajes sin desencriptar.

- **size_msg_cipher:** Tipo String.Tamaño total de los mensajes una vez encriptados.
- **ram_mv:** Tipo String. Valor de la RAM consumida en la máquina virtual para desencriptar mediante AES los mensajes de los bloques.
- **time_mv:** Tipo String. Tiempo empleado para desencriptar mediante AES los mensajes de los bloques en la máquina virtual.

4 Implementación del sistema

A lo largo de este capítulo, se explicará con detalles la funcionalidad de cada uno programas desarrollados para ambas soluciones. El proyecto desarrollado cuenta con la siguiente estructura presentada en la Figura 4-1.

El proyecto se divide en 3 carpetas principales:

- *Arduino* contiene los programas ejecutados en el ESP32. Cuenta con la carpeta *ESP32_WIFI* que contiene el programa que se ejecutará en la solución 2, esto es el ESP32_WIFI.ino.
- *MV-Azure* contiene los archivos que se emplearan en la máquina virtual (*cloud*) para las dos soluciones del sistema.
- *Rasp* dispone de los archivos necesarios para ejecutar en la Raspberry Pi (*fog*) tanto la solución 1 como la solución 2 del sistema.

Las diferentes extensiones de los archivos desarrollados son: Python (.py), Arduino (.ino) y Golang (.go).

En este capítulo se explican cada uno de los ficheros mostrados en esta figura a excepción de:

- El archivo *VS19-MV_key.pem*, en la carpeta Rasp, contiene la clave que permite conectar por SSH la Raspberry Pi con la máquina virtual en el *cloud*.
- Los archivos *go.mod* y *go.sum* contienen información que Golang utiliza para realizar un seguimiento de la configuración del módulo.

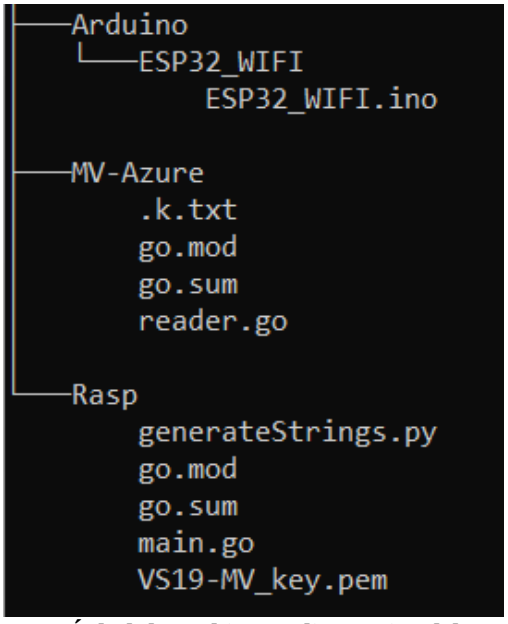

Figura 4-1 Árbol de archivos y directorios del proyecto

4.1 Implementación del ESP32

En esta sección se explicará cómo instalar el Arduino IDE, que es el entorno utilizado para la programación del ESP32, ya que permite acceder a bibliotecas estándar de gran utilidad. A continuación, se explicará el código fuente desarrollado en ESP32_WIFI.ino.

4.1.1 Instalación del IDE

Para la instalación del IDE se deben llevar a cabo los siguientes pasos:

1. Descargar el ejecutable directamente de la página oficial de Arduino[24]. Instalar el Arduino IDE.
2. Instalar la tarjeta ESP32, ya que no viene incluida por defecto en el entorno. Para ello, dirigirse al menú del IDE: *Archivo* > *Preferencias*. Copiar la siguiente URL en la barra *Gestor de URLs Adicionales de Tarjetas*: https://dl.espressif.com/dl/package_esp32_index.json. Véase la Figura 4-2.

   Nota: Si hay otra URL, antes de pegarla, colocar una coma al final y dar un espacio.

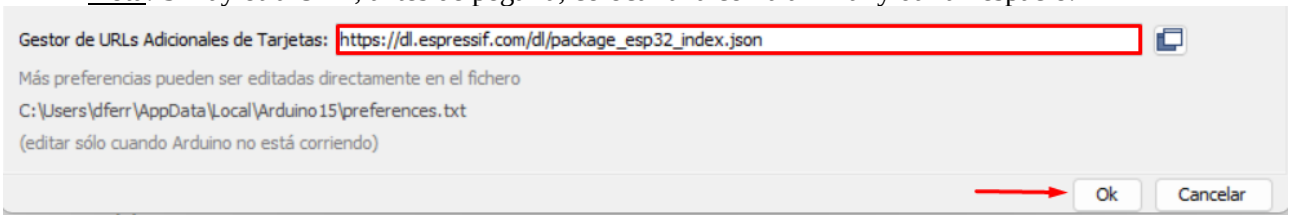

Figura 4-2 Gestor de URLs de tarjetas

3. Dar clic en Ok.
4. Clic en el IDE en *Herramientas* > *Placa* > *Gestor de tarjetas*
5. Escribir la palabra **esp32** para buscar e instalar la librería *esp32 by Espressif Systems*. Véase la Figura 4-3.

Figura 4-3 Librería esp32

Para comprobar la instalación de las tarjetas ESP32 en Arduino IDE, se deben llevar a cabo los siguientes pasos:

1. En el IDE, ir a la barra de Menú: *Herramientas* > *Placa* > *ESP32 Arduino* > *ESP32 Dev Module*.
2. Conectar la placa al PC.
3. Luego ir a *Herramientas* y seleccionar el puerto *COM* que le asignó el ordenador a la placa.
4. Para comprobar que todo fue bien, puede cargar el programa *Blink* en el ESP32 para encender y apagar un led. Este código se puede encontrar en el IDE en: *Archivo* > *Ejemplos* > *Basics* > *Blink*.

[24] https://www.arduino.cc/

4.1.2 Instalación de las librerías necesarias

Como se sabe, una librería o biblioteca es un archivo o conjunto de archivos que facilitan la programación, ya que aportan nuevas funcionalidades a los programas con el foco en la reutilización. Muchos desarrolladores crean librerías de forma altruista para poder comunicarse con sensores, otros circuitos integrados y otras placas diferentes para el entorno Arduino. Se diferencias dos tipos de librerías: estándar (desarrolladas por el equipo de Arduino) y no estándar (librerías desarrolladas por terceros).

En el caso de requerir incluir una librería estándar, la forma más sencilla de incluirla es mediante el Gestor de Librerías que se encuentra en *Programa* > *Incluir Librería* > *Administrar Bibliotecas* o el atajo de teclado: *Ctrl+Mayús+I*. Véase la Figura 4-4.

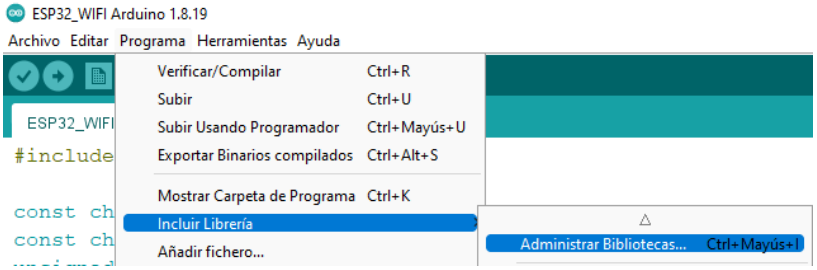

Figura 4-4 Incluir librería en Arduino IDE

Por ejemplo, en este proyecto se requiere del uso del módulo WiFi del ESP32, por lo que se necesita instalar la librería WiFi. Para ello, se escribe la palabra clave "Wifi*"* en la barra de búsqueda para filtrar. Escoger la última versión (1.2.7 en el momento en que se escribe) e *Instalar*. Solo se requiere de esta librería para el desarrollo de este proyecto.

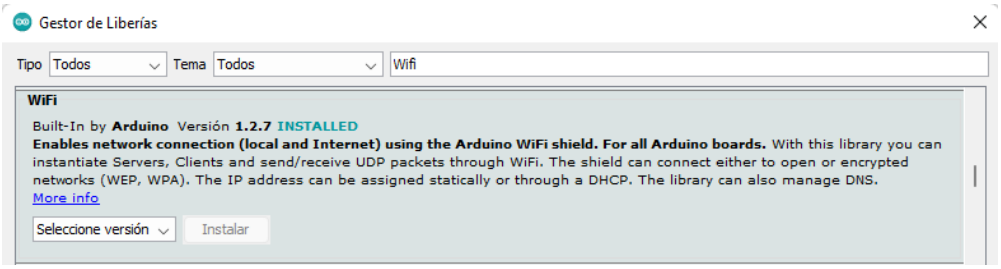

Figura 4-5 Instalar librería WiFi

4.1.3 Implementación del código fuente (ESP32_WIFI.ino)

Tal y como se ha introducido previamente, la solución 2 incluye un dispositivo ESP32, que mediante la red WiFi accesible a través de la librería *WiFi.h* permite enviar datos (cadenas de caracteres) a la Raspberry Pi. Para ello, se establece conexión por el puerto 9000 con el servidor TCP alojado en la Raspberry Pi.

A continuación, se detallará cada una de las funciones del archivo ESP32_WIFI.ino y se acompañará de una captura de código en las que se estimaron más relevantes.

Inicialización

Además de incluir la librería para el control de la WiFi, se requiere declarar/inicializar las siguientes variables (véase la Figura 4-6):

- *ssid* y *password*: nombre y la contraseña de la red WiFi a la que se desea conectar.
- *hostNombre*: Nombra al dispositivo ESP32.
- *serverIP*: Dirección IP del servidor, es decir, la Raspberry.
- *serverPort*: El número de puerto por donde el servidor estará escuchando.
- *client*: declara un objeto cliente para establecer la conexión con el servidor.

ESP32_WIFI

```
#include <WiFi.h>

const char* ssid = "MOVISTAR_D18A";
const char* password = "ULRR2WF5DosRAyD";

String hostNombre = "ESP32";
const IPAddress serverIP(192,168,0,36); //La dirección que desea visitar
uint16_t serverPort = 9000;
WiFiClient client; //Declarar un objeto cliente para conectarse al servidor
```

Figura 4-6 Inicio de las variables en el ESP32

setup()[25]

A continuación, véase la Figura 4-7, se debe llevar a cabo la inicialización de la comunicación serie, el módulo WiFi y el generador de números aleatorios. Nótese que el método setup() es ejecutado automáticamente, sin necesidad de llamada previa. Esto es:

- La transmisión serie se establece, de forma arbitraria, en 115200 baudios.
- Llama a el método initWiFi() para intentar conectar el dispositivo con el módulo WiFi. Se detallará un poco más adelante.
- randomSeed() inicializa el generador de números pseudoaleatorios, haciendo que comience en un punto arbitrario de su secuencia aleatoria.

Los siguientes eventos o interrupciones permiten que el ESP32 no esté constantemente revisando el estado de WiFi. Son muy útiles para detectar si la conexión se perdió o se restableció:

- *SYSTEM_EVENT_STA_CONNECTED*: el ESP32 está conectado en modo estación a un punto de acceso/router.
- *SYSTEM_EVENT_STA_DISCONNECTED*: la estación ESP32 desconectada del punto de acceso.

```
void setup()
{
  Serial.begin(115200);
  initWiFi();
  randomSeed(millis());
  WiFi.onEvent(WiFiStationConnected, SYSTEM_EVENT_STA_CONNECTED);
  WiFi.onEvent(WiFiStationDisconnected, SYSTEM_EVENT_STA_DISCONNECTED);
}
```

Figura 4-7 Función setup() del ESP32

[25] Se encarga de recoger la configuración y se invoca una sola vez, cuando el programa empieza.

initWiFi()
Nombra a el ESP32. Intenta conectarse a la red deseada. Si lo consigue, muestra: el nombre de la red, IP local del ESP32 y su dirección MAC. Véase la Figura 4-8.

```
void initWiFi()
{
  WiFi.mode(WIFI_STA); //el ESP32 se conecta a un punto de acceso
  WiFi.config(INADDR_NONE, INADDR_NONE, INADDR_NONE, INADDR_NONE);
  WiFi.setHostname(hostNombre.c_str()); //nombra el hostname

  Serial.print("Intentando conectarse a: ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(500);
  }
  Serial.println(); Serial.println("Conectado a: " + WiFi.SSID());
  Serial.print("IP: "); Serial.println(WiFi.localIP());
  Serial.println("macAddress: " + WiFi.macAddress());
}
```

Figura 4-8 Función initWifi() del ESP32

WiFiStationConnected(WiFiEvent_t event, WiFiEventInfo_t info)
Imprime por el Monitor Serie el mensaje "Conectado correctamente!". Véase la Figura 4-9.

```
void WiFiStationConnected(WiFiEvent_t event, WiFiEventInfo_t info)
{
  Serial.println("Conectado correctamente!");
}
```

Figura 4-9 Función WiFiStationConnected() del ESP32

WiFiStationDisconnected(WiFiEvent_t event, WiFiEventInfo_t info)
Muestra la razón de la desconexión e intenta conectarse nuevamente a la WiFi. Véase la Figura 4-10.

```
void WiFiStationDisconnected(WiFiEvent_t event, WiFiEventInfo_t info)
{
  Serial.print("Perdida de la conexión WiFi. Razón: ");
  Serial.println(info.disconnected.reason);
  Serial.println("Reconectando...");
  WiFi.begin(ssid, password);
}
```

Figura 4-10 Función WiFiStationDisconnected() del ESP32

loop()[26]
Si el ESP32 logra conectarse a la WiFi. Pueden pasar dos cosas:

1. El servidor se encuentra escuchando por el puerto especificado, esperando recibir un número aleatorio de cadenas alfanuméricas que serán enviadas.
2. No se puede establecer conexión con el servidor. Se muestra el mensaje de “Acceso fallido”.

En ambos casos, se detiene el cliente y se esperan 20 segundos entre cada ejecución del bucle. Véase la Figura 4-11. Nótese que el método loop() es ejecutado automáticamente de forma cíclica, sin necesidad de llamada previa.

[26] contienen el programa que se ejecutará cíclicamente.

```
void loop()
{
  if((WiFi.status() == WL_CONNECTED))
  {
    Serial.println("Intenta acceder al servidor...");
    if (client.connect(serverIP, serverPort))
    {
      Serial.println("Visita exitosa");
      sendStringsToServer(client , random(1,20));
    }else{
       Serial.println("Acceso fallido");
    }
    Serial.println("Cliente detenido");
    client.stop();
    delay(20000);
  }
}
```

Figura 4-11 Función loop() del ESP32

sendStringsToServer(WifiClient client, int numberStrings)
Recibe como parámetro el cliente y el número de cadenas a crear y enviar. En cada vuelta del bucle, la cadena devuelta por la función createString() es impresa por en la consola del IDE y despachada al servidor usando la coma como carácter separador. Por último, se envía un salto de línea (\n) al server para que sepa que es la última cadena que recibe, por el momento. Véase la Figura 4-12.

```
void sendStringsToServer(WiFiClient client ,int numberStrings)
{
  for(int i = 1; i <= numberStrings;i++)
  {
    String string = createString(20);
    Serial.println("Cadena: " + string);
    client.print(string);

    if(i != numberStrings) client.print(",");
  }
  client.print("\n");
}
```

Figura 4-12 Función sendStringsToServer() del ESP32

createString(int len)
La función devuelve una cadena alfanumérica del tamaño especificado por parámetro. Cada carácter de la cadena es seleccionado al azar del alfabeto (números, letras minúsculas y mayúsculas). Véase la Figura 4-13.

```
String createString(int len)
{
  String string = "";
  String alphabet [] =
    {"a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","u","v","w","x","y","z",
    "A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","U","V","W","X","Y","Z",
    "0","1","2","3","4","5","6","7","8","9"};

  int alphabetSize = sizeof(alphabet)/sizeof(alphabet[0]);
  for(int i = 0;i < len;i++)
    string += alphabet[random(alphabetSize)];
  return string;
}
```

Figura 4-13 Función createString() del ESP32

4.2 Implementación de la Raspberry Pi

Tal y como se ha introducido previamente, la Raspberry Pi interviene tanto en la solución 1 como en la solución 2. En la solución 1, un programa en Python generada y escribe las cadenas de caracteres en un archivo de texto y por su parte la solución 2, un servidor TCP recibe las cadenas de texto enviadas por el ESP32. En ambas soluciones, estas cadenas de texto, entre otros campos, conforman el contenido de los bloques que son enviados por SSH a la MV ubicada en el *cloud*.

A continuación, se detallará como instalar el sistema operativo en la Raspberry Pi. Además, las funciones de los archivos generateStrings.py y main.go son explicadas y acompañadas de sus respectivas capturas.

4.2.1 Instalación del sistema operativo

Como se ha introducido previamente, la Raspberry Pi será utilizada con el sistema operativo Raspbian, que debe ser cargado en una tarjeta SD mediante el uso de la herramienta *Raspberry Pi Imager*. Para ello, se requiere de los siguientes pasos:

- Formatear la tarjeta microSD. En este caso se ha optado por utilizar una tarjeta microSD de SanDisk con 64 GB con hasta 160 MB/s de velocidad de lectura y 60 MB/s de escritura.
- Abrir *Raspberry Pi Imager*.
- En la herramienta, clicar en *CHOOSE OS* para elegir el sistema operativo. Se optó por el Raspberry Pi OS (64-bit) ya que viene incorporado con Escritorio, es una versión completa compatible con la Raspberry Pi 3/4/400.
- Seleccionar la tarjeta microSD en *CHOOSE STORAGE*.
- Clicar en *WRITE* para graba el OS en la tarjeta.
- Retirar de forma segura la tarjeta.

A partir de este punto, la Raspberry estará lista para implementar las funcionalidades requeridas. Nótese que tanto Python (en su versión 3.10) como Go (en su versión Goland 15.5) se encuentran incluidos por defecto en Raspbian.

En aplicaciones donde se requiere la intercomunicación de dispositivos mediante direcciones IP, como en este desarrollo, es importante decidir el uso o bien de direcciones IP fijas, o bien dinámicas. En este caso concreto, se ha optado por utilizar una configuración estática de la interfaz de red WiFi de la Raspberry. Para ello, se debe modificar el archivo */etc/dhcpcd.conf* con el comando *sudo nano /etc/dhcpcd.*conf y añadiendo el siguiente contenido:

- interface wlan0
- static ip_address=192.168.1.200/24
- static routers=192.168.1.1
- static domain_name_servers=192.168.1.1 8.8.8.8

Donde,

- interface = Nombre de la interface que se quiere configurar
- static ip_address = Dirección fija que se quiere (dejar el /24 al final)
- static routers = Dirección del gateweay (del router)
- static domain_name_servers = Dirección del servidor DNS (normalmente la del router, o unas externas como las de Google 8.8.8.8). Si se quiere más de un servidor DNS se puede añadir separados por un espacio.

Finalmente, se guardan los cambios en el fichero y reinicia la Raspberry con el comando *sudo reboot*. Después del reinicio puede comprobarse mediante el comando *ifconfig wlan0* si la interfaz se configuró con la dirección IP deseada.

4.2.2 Implementación del código fuente generateStrings.py

El programa **generateStrings.py**, implementado en Python, nace con el objetivo de implementar un generador de cadenas alfanuméricas para la solución 1, como si éstas fueran generadas por el ESP32. Recuérdese que la solución 1 no incluye el ESP32, cuya principal función es la generación es tales cadenas de caracteres. Para ello, generateStrings.py genera un conjunto de cadenas alfanuméricas (de 1 a 20) cada 10 segundos, que son escritas en el archivo de texto strings.txt, véase en la Figura 4-14, por ejemplo, el contenido del fichero después de generar 5 cadenas aleatorias. El contenido de este fichero será enviado al *cloud*, transformado previamente mediante *Blockchain*, tal y como se verá más adelante.

strings.txt

```
1  rrbQxEW4Smsmei5rPXoq
2  WjdMpMuo60pG5sYYl50X
3  WwoExUA4akywGgwi7ptd
4  OvYWjfaOl9KT52G1FqEG
5  NpOya6SiUFOk2TNq8DEA
6
```

Figura 4-14 Cadenas en el archivo strings.txt

A continuación, se detalla cada uno de los procedimientos involucrados:

main()

Programa principal de este fichero de código, que permite actualizar el contenido del fichero strings.txt cada 20 segundos. Véase el código en la Figura 4-15.

```
def main():
    while True:
        updateStringsFile()
        sleep(20)
main()
```

Figura 4-15 Función main() del generateStrings.py

updateStringsFile()

Sobreescribe en el fichero strings.txt un conjunto aleatorio de cadenas alfanuméricas generadas (1-20) de tamaño 20. Véase el código en la Figura 4-16. Las cadenas generadas se muestran por consola.

```
def updateStringsFile():
    stringsFile = open('strings.txt','w')
    num = random.randint(1, 20); len = 20
    print("Generando ", num , " cadenas...")
    for _ in range(num):
        cadena = createStrings(len)
        stringsFile.write(cadena)
        print(cadena)
    stringsFile.close()
```

Figura 4-16 Función updateStringsFile() del generateStrings.py

createStrings(len)

Devuelve una cadena alfanumérica aleatoria del tamaño del valor pasado por parámetro junto con un salto de línea. Su implementación es exactamente la misma a la ya mostrada en la Figura 4-13 para el ESP32, pero en este caso bajo Python. Véase la Figura 4-17.

```
def createStrings(len):
    return ''.join(random.choices(string.ascii_uppercase + string.ascii_lowercase + string.digits, k=len)) + "\n"
```

Figura 4-17 Función createStrings() del generateStrings.py

4.2.3 Implementación del código fuente main.go

El programa **main.go**, implementado en Go, tomará las cadenas del strings.txt, las cifrará y generará un mensaje en forma de un bloque *Blockchain*, para ser enviado al *cloud*. Nótese que la codificación de la cadena de bloques de *Blockchain* es propia, pero está inspirada en la solución propuesta en [37].

Para llevar a cabo esta codificación se requiere de la importación de las bibliotecas presentes en el código expuesto en la Figura 4-18 y que detallan a continuación:

- **bufio:** realiza operaciones de E/S almacenadas en búfer.
- **crypto/rand:** generador de números pseudoaleatorios.
- **encoding/base64**: codifica en base64.
- **io/ioutil:** lleva a cabo algunas funciones de utilidad de E/S.
- **math/big:** aritmética de precisión arbitraria (números grandes).
- **net**: proporciona una interfaz portátil para E/S de red, que incluye TCP/IP, UDP, resolución de nombres de dominio y sockets de dominio Unix.
- **os:** proporciona una interfaz independiente de la plataforma para la funcionalidad del sistema operativo.
- **strings:** implementa funciones simples para manipular cadenas codificadas en UTF-8.
- **github.com/appleboy/easyssh-proxy:** incorpora a Go algunas características del protocolo SSH.
- **github.com/DmitriyVTitov/size:** mide el tamaño del objeto en tiempo de ejecución.

```
import (
    "bufio"
    "crypto/aes"
    "crypto/cipher"
    "crypto/rand"
    "crypto/sha256"
    "encoding/base64"
    "encoding/hex"
    "encoding/json"
    "fmt"
    "io"
    "io/ioutil"
    "log"
    "math/big"
    "net"
    "os"
    "runtime"
    "strconv"
    "strings"
    "time"

    "github.com/DmitriyVTitov/size"
    "github.com/appleboy/easyssh-proxy"
    "github.com/davecgh/go-spew/spew"
)
```

Figura 4-18 Librerías del main.go

A continuación, se detallan aquellos aspectos más relevantes de la implementación llevada a cabo.

Inicialización

El primer paso consiste en la inicialización de las estructuras de datos. Véase la Figura 4-19. A continuación, se proporciona el detalle de cada variable:

- *var client net.Conn***:** guardará el cliente que se conecte en la Solución 2.
- *var solution int*: guarda el identificador de la solución en uso, 1 o 2.
- *var key []byte*: clave para cifrar y descifrar las cadenas almacenados en los bloques, esto es, el mensaje.

- *Blockchain []Block*: array de bloques. A su vez, el tipo Block, véase la Figura 4-20, contiene la siguiente información:
  - *Index int*: posición del bloque en la cadena de bloques.
  - *Time_Stamp string*: fecha y hora de creación.
  - *MSG []string*: mensaje guardado (cadenas alfanúmericas).
  - *Hash string*: valor de hash del bloque actual.
  - *Prev_Hash string*: Hash del bloque anterior.
  - *Statistic Statistics*: Estructura en la que se guardaran un conjunto de valores estadísticos sobre la encriptación y desencriptación del mensaje guardado en el bloque. A su vez, la estructura Statistics contiene la siguiente información:
    - *RAM_Rasp string*: RAM empleada para encriptar el mensaje.
    - *Time_Rasp string*: tiempo tardado en encriptar.
    - *Size_MSG string*: tamaño del mensaje sin encriptar.
    - *Size_MSG_Cipher string*: tamaño del mensaje encriptado.
    - *RAM_MV string*: RAM empleada para desencriptar el mensaje.
    - *Time_MV string*: tiempo tardado en desencriptar.

```
var Blockchain []Block
var client net.Conn
var solution int
var key []byte
```

Figura 4-19 Inicio de las variables del main.go

```
type Block struct {
    Index      int
    Time_Stamp string
    MSG        []string
    Hash       string
    Prev_Hash  string
    Statistic  Statistics
}

type Statistics struct {
    RAM_Rasp        string
    Time_Rasp       string
    Size_MSG        string
    Size_MSG_Cipher string
    RAM_MV          string
    Time_MV         string
}
```

Figura 4-20 Estructura del bloque del main.go

main()

En el programa principal del código. Primeramente, se genera una clave (*key*) de encriptación y el bloque inicial de la cadena de bloques. Estos son escritos en .k.txt y data.json, respectivamente. Estos dos archivos son enviados por SSH a la máquina virtual. El programa puede recibir un valor por parámetro para determinar la solución por usar. Si el valor es 1 se ejecutará la función notServer(), diseñada para la solución 1, mientras que si el valor es 2, se ejecutará la función server(), diseñada para la solución 2. Véase la Figura 4-21.

```go
func main() {

    key = generateKey()
    sendFileServer(".k.txt", "/home/linux-vm/TFG/")

    createBlockInitial()

    updateJSON(Blockchain, "data.json")
    sendFileServer("data.json", "/home/linux-vm/TFG/")

    solution = 1
    if len(os.Args) == 2 {
        arg, err := strconv.Atoi(os.Args[1])
        if err != nil {
            panic(err)
        }

        switch arg {
        case 1:
            notServer()
        case 2:
            solution = 2
            server()
        default:
            fmt.Println("go run main.go (1 ó 2)")
        }

    } else {
        notServer()
    }

}
```

Figura 4-21 Función main() del main.go

generateKey() []byte

Genera una clave de forma aleatoria cada vez que se ejecuta el programa, por motivos de seguridad. Esta clave se escribe en el archivo .k.txt y es retornada por la función. Véase la Figura 4-22.

```go
func generateKey() []byte {

    str := ""
    for i := 0; i < 22; i++ {
        r, err := rand.Int(rand.Reader, big.NewInt(10))
        if err != nil {
            log.Fatal(err)
        }
        str = str + strconv.FormatInt(r.Int64(), 10)
    }

    stre := base64.StdEncoding.EncodeToString([]byte(str))
    key := []byte(stre)
    ioutil.WriteFile(".k.txt", key, 0777)
    return key
}
```

Figura 4-22 Función generateKey() del main.go

sendFileServer(file string, path string)

Copia el archivo pasado por parámetro a la ruta especificada de la máquina virtual de Azure. La conexión se establece por SSH y se copia con SCP. Muestra un mensaje indicando si el archivo se ha enviado o no. Véase Figura 4-23.

```
func sendFileServer(file string, path string) {

    ssh := &easyssh.MakeConfig{
        User:    "linux-vm",
        Server:  "20.199.10.61",
        KeyPath: "./VS19-MV_key.pem", //tiene que tener el permido chmod 400
        Port:    "22",
    }

    err := ssh.Scp(file, path+file)
    if err != nil {
        panic("No se pudo enviar el archivo: " + file)
    } else {
        fmt.Println("Archivo " + file + " enviado")
    }
}
```

Figura 4-23 Función sendFileServer del main.go

createBlockInitial()

Crea y añade un bloque inicial al *Blockchain*. Este bloque tendrá el Index 0, el correspondiente Timestamp y su mensaje (campo MSG) será la palabra "Inicial". El resto de los valores del bloque no se rellenan. Véase Figura 4-24.

```
func createBlockInitial() {

    t := time.Now()
    MSG := encrypt("Inicial")
    genesisBlock := Block{0, t.String(), []string{MSG}, "", "", Statistics{"", "", "", "", "", ""}}
    spew.Dump(genesisBlock)
    Blockchain = append(Blockchain, genesisBlock)
}
```

Figura 4-24 Función createBlockInitial() del main.go

updateJSON(Blocks []Block, file string)

Escribe la cadena de bloques en el archivo indicado por parámetro (data.json) que es un archivo que almacena los bloques de la cadena actual (solo una cadena). Véase Figura 4-25.

```
func updateJSON(Blocks []Block, file string) {

    output, err := json.Marshal(Blocks)
    if err != nil {
        log.Fatal(err)
    }
    ioutil.WriteFile(file, output, os.ModePerm)
}
```

Figura 4-25 Función updateJSON() del main.go

notServer()

Cada 20 segundos llama a la función generateBlock(). Dicha función se ejecuta concurrentemente gracias a la palabra reservada *go*. Véase la Figura 4-26.

```
func notServer() {

    for {
        go generateBlock()
        time.Sleep(20 * time.Second)
    }
}
```

Figura 4-26 Función notServer() del main.go

server()

Crea un servidor TCP que estará escuchando por el puerto 9000 para aceptar la conexión a todo cliente que se conecte, en este caso el nodo ESP32. Véase la Figura 4-27.

El protocolo TCP considerado proporciona una transmisión confiable mediante el reenvío de paquetes perdidos, garantizando que lleguen su destino en el mismo orden que se transfirieron y sin errores. Condiciones necesarias e imprescindibles para el envío de los datos en esta solución. En cambio, el UDP es más rápido que el TCP, pero no garantiza la entrega de los datos ni verifica los errores.

```
func server() {

    server, err := net.Listen("tcp", ":9000")
    fmt.Println("El servidor esta creado")

    if err != nil {
        panic(err)
    }
    defer server.Close()

    for {
        fmt.Println("Esperando un cliente....")
        client, err = server.Accept()
        if err != nil {
            panic(err)
        }
        fmt.Println("Un cliente se conecto")

        go generateBlock()
    }
}
```

Figura 4-27 Función server() del main.go

generateBlock()

La solución por utilizar determinará la forma en el que se genere el nuevo bloque. En la Solución 1 se llama a readFileString(), mientras que en la solución 2 se llama a readStringsClient(). Esto es, en la solución 1 se realiza una lectura de fichero, mientras que en la solución 2 se espera la recepción de información por parte del ESP32. En todo caso, si el nuevo bloque es válido, se añade a la cadena de bloques actual (*Blockchain*), se muestra por consola y se sobrescribe el contenido del archivo newBlock.json. Este JSON se copia en la ruta indicada de la máquina virtual. El data.json es actualizado.

Nota: Nótese que newBlock.json contiene un único bloque cada vez, mientras que data.json contiene la cadena completa. Véase la Figura 4-28.

```
func generateBlock() {

    str := []string{}
    if solution == 1 {
        str = readFileStrings()
    } else {
        str = readStringsClient(client)
    }

    newBlock := createBlock(Blockchain[len(Blockchain)-1], str)

    if isBlockValid(newBlock, Blockchain[len(Blockchain)-1]) {
        Blockchain = append(Blockchain, newBlock)
        spew.Dump(newBlock)

        updateJSON([]Block{newBlock}, "newBlock.json")
        sendFileServer("newBlock.json", "/home/linux-vm/TFG/")
        updateJSON(Blockchain, "data.json")
    }
}
```

Figura 4-28 Función generateBlock() del main.go

readFileStrings() []string

Lee el archivo strings.txt, generado en la Raspberry, por líneas. Extrae la cadena de cada línea para insertarla en un array. Una vez leído todo el archivo, se retorna dicho array. Véase Figura 4-29.

```
func readFileStrings() []string {

    readFile, err := os.Open("strings.txt")
    if err != nil {
        panic(err)
    }

    fileScanner := bufio.NewScanner(readFile)

    strings := []string{}
    for fileScanner.Scan() {
        strings = append(strings, fileScanner.Text())
    }
    readFile.Close()
    return strings
}
```

Figura 4-29 Función readFileStrings() del main.go

readStringsClient(client net.Conn) []string

El servidor leerá todas las cadenas, separadas por comas, que envíe el cliente, el nodo ESP32, hasta que encuentre un salto de línea '\n'. Se devolverá un array con todas las cadenas recibidas. Véase Figura 4-30.

```
func readStringsClient(client net.Conn) []string {
    defer client.Close()

    netData, err := bufio.NewReader(client).ReadString('\n')
    if err != nil {
        panic(err)
    }

    strings := strings.SplitAfter(strings.Replace(netData, "\n", "", -1), ",")
    return strings
}
```

Figura 4-30 Función readStringsClient() del main.go

isBlockValid(newBlock, oldBlock Block) bool

Devuelve un booleano indicando la validez del bloque. Véase Figura 4-31. Para que un bloque sea válido, se tienen que cumplir estas tres condiciones:

- El índice del nuevo bloque es consecutivo al índice del último bloque.
- El hash del último bloque debe ser igual al PreHash del bloque nuevo.
- El hash del bloque nuevo tiene que ser equivalente al hash calculado.

```
func isBlockValid(newBlock, oldBlock Block) bool {

    if oldBlock.Index+1 != newBlock.Index {
        fmt.Println("Old Index: ", oldBlock.Index)
        fmt.Println("New Index: ", newBlock.Index)
        fmt.Println("Index problem")
        return false
    }

    if oldBlock.Hash != newBlock.Prev_Hash {
        fmt.Println("Old Hash: ", oldBlock.Hash)
        fmt.Println("New PrevHash: ", newBlock.Prev_Hash)
        fmt.Println("PrevHash problem")
        return false
    }

    if calculateHash(newBlock) != newBlock.Hash {
        fmt.Println("Calculated: ", calculateHash(newBlock))
        fmt.Println("Stored: ", newBlock.Hash)
        fmt.Println("CalculateHash problem")
        return false
    }

    return true
}
```

Figura 4-31 Función isBlockValid() del maingo

createBlock(oldBlock Block, MSG string) (Block, error)

Crea y retorna un nuevo bloque a partir del índice y el hash del último bloque introducido en la cadena de bloques. En el proceso de creación, se toman los siguientes valores estadísticos: RAM_Rasp, SizeMSG, TimeRasp y SizeMSGCipher. Véase la Figura 4-32.

```
func createBlock(oldBlock Block, str []string) Block {

    var newBlock Block

    newBlock.Index = oldBlock.Index + 1
    t := time.Now()
    newBlock.Time_Stamp = t.String()

    newBlock.Statistic.Size_MSG = strconv.Itoa(size.Of(str)) + " bytes"
    timeBefore := time.Now()
    memBefore := memUsage()
    for i := 0; i < len(str); i++ {
        encrypString := encrypt(str[i])
        str[i] = string(encrypString)
    }
    memNow := memUsage()

    newBlock.Statistic.RAM_Rasp = strconv.Itoa(int((memNow - memBefore))) + " kB"
    newBlock.Statistic.Time_Rasp = time.Since(timeBefore).String()
    newBlock.Statistic.Size_MSG_Cipher = strconv.Itoa(size.Of(str)) + " bytes"

    newBlock.MSG = str
    newBlock.Prev_Hash = oldBlock.Hash
    newBlock.Hash = calculateHash(newBlock)

    return newBlock
}
```

Figura 4-32 Función createBlock() del main.go

encrypt(message string) (encoded string)

Recibe por parámetro una cadena de caracteres y retorna la cadena cifrada por el algoritmo AES que necesitará la *key* generada antes. Véase la Figura 4-33.

```
func encrypt(message string) (encoded string) {

    //Create a new AES cipher using the key
    block, err := aes.NewCipher(key)
    if err != nil {
        return
    }

    //Make the cipher text a byte array of size BlockSize + the length of the message
    cipherText := make([]byte, aes.BlockSize+len([]byte(message)))

    //iv is the ciphertext up to the blocksize (16)
    iv := cipherText[:aes.BlockSize]
    if _, err = io.ReadFull(rand.Reader, iv); err != nil {
        return
    }

    //Encrypt the data:
    stream := cipher.NewCFBEncrypter(block, iv)
    stream.XORKeyStream(cipherText[aes.BlockSize:], []byte(message))

    //Return string encoded in base64
    return base64.RawStdEncoding.EncodeToString(cipherText)
}
```

Figura 4-33 Función encrypt() del main.go

memUsage() uint64

Retorna la cantidad de kB de memoria utilizados en tiempo de ejecución. Véase la Figura 4-34.

```
func memUsage() uint64 {

    var m runtime.MemStats
    runtime.ReadMemStats(&m)

    RAM := m.Alloc / 1024
    return RAM
}
```

Figura 4-34 Función memUsage() del main.go

calculateHash(block Block) string

Calcula y retorna un hash único teniendo en cuenta el Index, Timestamp, todas las cadenas del MSG y el PrevHash del bloque actual. Véase la Figura 4-35.

```
func calculateHash(block Block) string {

    var allMSG string
    for i := 0; i < len(block.MSG); i++ {
        allMSG += block.MSG[i]
    }

    record := string(block.Index) + block.Time_Stamp + allMSG + block.Prev_Hash
    h := sha256.New()
    h.Write([]byte(record))
    hashed := h.Sum(nil)
    return hex.EncodeToString(hashed)
}
```

Figura 4-35 Función calculateHash() del main.go

4.3 Implementación del cloud

Para la implementación del *cloud*, se ha optado por utilizar una máquina virtual de Azure, las cuales pueden crearse mediante el Azure Portal. Azure Portal es una interfaz de usuario basada en explorador para crear recursos de Azure. Para implementar una máquina virtual Linux en Azure que ejecute Ubuntu 20.04 LTS se debe tener una suscripción en Azure. La disponibilidad de los servicios dependerá del tipo de cuenta, (ejemplo estudiante o empresa). Por lo general, Azure ofrece un conjunto de servicios populares de forma gratuita durante 12 meses, más de 40 servicios gratis para siempre y un crédito de 200$ para gastar en 30 días. Cuando se gaste el tiempo o el crédito gratis se empezará a pagar por uso [38].

La máquina virtual del proyecto fue creada en Azure Portal con un plan de pago en el que disponía de 200$/mes durante 1 año. Los enlaces consultados para la creación y visualización de esta máquina virtual fueron los siguientes:

- Creación de una máquina virtual Linux en Azure Portal: https://docs.microsoft.com/es-es/azure/virtual-machines/linux/quick-create-portal
- Instalación y configuración de xrdp para usar Escritorio remoto con Ubuntu: https://docs.microsoft.com/es-es/azure/virtual-machines/linux/use-remote-desktop?tabs=azure-cli

Los enlaces consultados para la creación de la BD en *MongoDB* y la instalación de *MongoDB Compass* fueron los siguientes:

- Instalación de *MongoDB* en Ubuntu. Los datos estarán almacenados en la propia máquina virtual, siendo su dirección la 127.0.0.1, por lo que solo puede aceptar conexiones de clientes que se ejecutan en la misma máquina: https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/
- Instalación del *MongoDB Compass*. Esta herramienta proporciona una interfaz de usuario para consultar, optimizar y analizar los datos en *MongoDB*: https://www.mongodb.com/docs/compass/current/install/

A continuación, se proporcionan los detalles más importantes de la implementación.

4.3.1 Implementación del código fuente reader.go

El programa **reader.go**, implementado en Go, descifra las cadenas de los bloques enviados por la Raspberry Pi, se muestra el contenido de los bloques por la consola de la máquina virtual, para luego escribirlo en la BD.

Para llevar a cabo esta codificación se requiere de la importación de las bibliotecas presentes en el código expuesto en la Figura 4-36 y que detallan a continuación:

- **crypto/aes**: ejecuta el cifrado AES.

- **crypto/cipher**: contiene algoritmos para proteger la confidencialidad de datos.
- **fmt**: formatea las E/S. Posee funciones análogas a printf y scanf de C.
- **crypto/sha256**: activa los algoritmos hash SHA224 y SHA256.
- **encoding/hex**: implementa la codificación y decodificación hexadecimal.
- **enconding/json**: codifica y decodifica los archivos JSON.
- **log**: define un tipo, Logger, con métodos para dar formato a la salida.
- **time**: permite medir y mostrar el tiempo.
- **errors**: implementan funciones para manipular errores.
- **runtime**: permite a un programador controlar el comportamiento del tiempo de ejecución
- **strconv**: convierte hacia y desde representaciones de cadenas de tipos de datos básicos.
- **runtime**: permite a un programador controlar el comportamiento del tiempo de ejecución.
- **github.com/davecgh/go-spew/spew**: se empleó para imprimir la cadena de bloques por consola.
- **go.mongodb.org/mongo-driver/***: proporcionan una API de controlador *MongoDB* para Go.

```
import (
    "context"
    "crypto/aes"
    "crypto/cipher"
    "crypto/sha256"
    "encoding/base64"
    "encoding/hex"
    "encoding/json"
    "errors"
    "fmt"
    "io/ioutil"
    "log"
    "os"
    "runtime"
    "strconv"
    "time"

    "github.com/davecgh/go-spew/spew"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
    "go.mongodb.org/mongo-driver/mongo/readpref"
)
```

Figura 4-36 Librerías del reader.go

A continuación, se detallan aquellos aspectos más relevantes de la implementación llevada a cabo. Nótese que las funciones isBlockValid()**,** calculateHash()**,** updateJSON() **y** memUsage() son las mismas que las ya explicadas para el programa main.go en la Sección 4.2.3

Inicialización

El primer paso consiste en la inicialización de las estructuras de datos. Véase la Figura 4-37. A continuación, se proporciona el detalle de cada variable:

- *var Blockchain []Block*: Crea el array de cadenas de bloques.
- *var LastBlock Block*: guarda el último bloque introducido en la cadena de bloques.
- *var collectionMG *mongo.Collection*: crea una colección de *MongoDB*.

```
var Blockchain []Block
var LastBlock Block
var collectionMG *mongo.Collection
```

Figura 4-37 Inicio de las variables del reader.go

Nótese que la definición de Block es la misma que la ya vista en main.go en la Sección 4.2.3.

main()

Es el procedimiento principal en reader.go. En este código, la función getCollection() permite acceder la colección "*Blockchain*" de la Base de Datos "db_TFG" del servidor *MongoDB* en la máquina virtual. Se espera a recibir una nueva clave de cifrado que será escrita en el archivo .k.txt. Se esperan los archivos data.json y newBlock.json con el bloque inicial y el primer bloque de la cadena respectivamente. Estos dos bloques se añaden a la cadena de bloques actual y se escriben en data.json (se emplea para guardar todos los bloques que componen la cadena actual). Véase la Figura 4-38.

```
collectionMG = getCollection("Blockchain")
waitChangeKey(".k.txt")
// read temporal key
key, err := ioutil.ReadFile(".k.txt")
if err != nil {
    log.Fatal(err)
}

initialBlock := waitBlock("data.json", 0)
block := waitBlock("newBlock.json", 1)

addToBlockchain(initialBlock, key)
addToBlockchain(block, key)

updateJSON(Blockchain, "data.json")
```

Figura 4-38 Parte 1 de la función main() del reader.go

En la Figura 4-39 continúa la explicación del método main(). En este código, cada 10 segundos, el fichero newBlock.json es leído, si hay un nuevo bloque (el hash es distinto al del último bloque leído) y es este es válido, se añade a la cadena de bloque y se actualiza el archivo data.json. Si el bloque no es válido, se muestra un mensaje de error "Bloque incorrecto." y el programa se detiene.

```
for {
    time.Sleep(10 * time.Second)
    block := getBlocks("newBlock.json")[0]

    if LastBlock.Hash != block.Hash {

        if isBlockValid(block, LastBlock) {

            addToBlockchain(block, key)
            updateJSON(Blockchain, "data.json")

        } else {
            err := errors.New("Bloque incorrecto")
            panic(err)
        }
    }
}
```

Figura 4-39 Parte 2 de la función main() del reader.go

getCollection(collection string) *mongo.Collection

Crea una conexión con el servidor local, en la máquina virtual de Azure, de *MongoDB* por el puerto 27017. Hace un ping al servidor de *MongoDB* para comprobar que se encuentra activo. En tal caso, devuelve la referencia a la base de datos "db_TFG" y su colección "*Blockchain*". Véase la Figura 4-40.

```
func getCollection(collection string) *mongo.Collection {

    client, err := mongo.Connect(context.TODO(), options.Client().ApplyURI("mongodb://localhost:27017/"))
    if err != nil {
        panic(err)
    }
    ctx, _ := context.WithTimeout(context.Background(), 10*time.Second)
    err = client.Connect(ctx)

    if err := client.Ping(context.TODO(), readpref.Primary()); err != nil {
        panic(err)
    }

    return client.Database("db_TFG").Collection(collection)
}
```

Figura 4-40 Función getCollection() del reader.go

waitChangeKey(file string)

Espera a que se reciba la nueva clave para descifrar enviada desde la Raspberry Pi. Como es de esperar, este envío es crítico y puede llevar a comprometer la seguridad si se comprometiera la clave. En este código, se guarda la fecha de la última modificación del archivo .k.txt. Hay un bucle que comprueba cada 5 segundos si este archivo ha sido modificado, cuando lo sea , el bucle se rompe y se continua con la ejecución del programa Véase la Figura 4-41.

Nota: Cada nueva ejecución, se envía el archivo .k.txt por SSH de la Raspberry Pi a la máquina virtual. Si no se espera por la nueva clave, el programa cogerá la clave antigua que está en el .k.txt e intentar descifrar el bloque con ésta y conducirá a error ya que fue cifrado con otra clave.

```
func waitChangeKey(file string) {

    modifiedtime := getModificationDate(file)
    fileModified := false

    for !fileModified {
        time.Sleep(5 * time.Second)

        modTime := getModificationDate(file)
        if modTime.After(modifiedtime) {
            modifiedtime = modTime
            fileModified = true
        }
    }

}
```

Figura 4-41 Función waitChangeKey() del reader.go

getModificationDate(name string) time.Time

Obtiene la fecha de modificación del archivo pasado por parámetro (.k.txt.). Véase la Figura 4-42.

```go
func getModificationDate(name string) time.Time {
    file, err := os.Stat(name)
    if err != nil {
        fmt.Println(err)
    }
    return file.ModTime()
}
```

Figura 4-42 Función getModificationDate() del reader.go

waitBlock(file string, blockIndex int) Block

Esta función retorna el bloque cuyo índice es blockIndex y que se encuentra en el fichero file. Véase la Figura 4-43.

```go
func waitBlock(file string, blockIndex int) Block {

    blocks := []Block{}
    correctBlock := false

    for !correctBlock {
        blocks = getBlocks(file)
        if len(blocks) == 1 && blocks[0].Index == blockIndex {
            correctBlock = true
        }
    }
    return blocks[0]
}
```

Figura 4-43 Función waitBlock() del reader.go

getBlocks(file string) []Block

Lee el contenido del fichero introducido por parámetro (file) y devuelve un array de bloques con el bloque guardado. Véase la Figura 4-44.

```go
func getBlocks(file string) []Block {

    archivo, err := ioutil.ReadFile(file)
    if err != nil {
        log.Fatal(err)
    }
    var b []Block
    err = json.Unmarshal(archivo, &b)
    if err != nil {
        log.Fatal(err)
    }
    return b
}
```

Figura 4-44 Función getBlocks() del reader.go

addToBlockchain(block Block, key []byte)

Como se puede apreciar en la Figura 4-45, el bloque pasado por parámetro (block) es insertado al *Blockchain*, convirtiéndose en el nuevo último bloque de la cadena, es decir, el LastBlock. Para ello, en primer lugar, el bloque se desencripta mediante la función decryptBlock(), a continuación, se inserta en la base de datos *MongoDB* con insertBlock y se muestra el contenido del bloque por la consola de la máquina virtual.

```
func addToBlockchain(block Block, key []byte) {

    newBlock := decryptBlock(block, key)
    insertBlock(newBlock)
    spew.Dump(newBlock)
    Blockchain = append(Blockchain, newBlock)
    LastBlock = newBlock
}
```

Figura 4-45 Función addToBlockchain() del reader.go

decryptBlock(block Block, key []byte) Block

Con la clave obtenida previamente (*key*), se desencripta el mensaje y se muestra el contenido del bloque por consola. El valor de MSG es actualizado por el texto descifrado y se retorna el bloque. Véase la Figura 4-46.

```
func decryptBlock(block Block, key []byte) Block {

    timeBefore := time.Now()
    memBefore := memUsage()
    for i := 0; i < len(block.MSG); i++ {

        decryptStrins := decrypt(key, block.MSG[i])
        block.MSG[i] = decryptStrins
    }
    memNow := memUsage()
    block.Statistic.RAM_MV = strconv.Itoa(int((memNow - memBefore))) + " kB"
    block.Statistic.Time_MV = time.Since(timeBefore).String()

    return block
}
```

Figura 4-46 Función decryptBlock() del reader.go

decrypt(key []byte, secure string) (decoded string)

La clave y la cadena encriptada son recibidos por parámetros. La función retorna la cadena descifrada por el algoritmo AES y la clave. Véase la Figura 4-47.

```
func decrypt(key []byte, secure string) (decoded string) {
    //Remove base64 encoding:
    cipherText, err := base64.RawStdEncoding.DecodeString(secure)

    //IF DecodeString failed, exit:
    if err != nil {
        return
    }

    //Create a new AES cipher with the key and encrypted message
    block, err := aes.NewCipher(key)

    //IF NewCipher failed, exit:
    if err != nil {
        return
    }

    iv := cipherText[:aes.BlockSize]
    cipherText = cipherText[aes.BlockSize:]

    //Decrypt the message
    stream := cipher.NewCFBDecrypter(block, iv)
    stream.XORKeyStream(cipherText, cipherText)

    return string(cipherText)
}
```

Figura 4-47 Función decrypt() del reader.go

insertBlock(bloque Block)

El bloque proporcionado por parámetro (block) es grabado en la colección de *MongoDB* y su id es mostrado por consola. Véase la Figura 4-48.

```
func insertBlock(bloque Block) {

    result, err := collectionMG.InsertOne(context.TODO(), bloque)
    if err != nil {
        panic(err)
    }
    // display the id of the newly inserted object
    fmt.Println(result.InsertedID)
}
```

Figura 4-48 Función insertBlock() del reader.go

5 Experimentación

En este capítulo se detallan ciertos aspectos relacionados con la experimentación. Por un lado, se muestra un ejemplo de ejecución del sistema, así como un análisis práctico de la seguridad proporcionada. A continuación, se muestran y analizan una serie de tablas y gráficas fruto de un estudio experimental donde se estudia cómo afecta a ciertos parámetros del sistema el tamaño del mensaje enviado.

5.1 Puesta en marcha y seguridad

A continuación, se proporciona un ejemplo práctico de ejecución del sistema para comprender el tipo de respuesta obtenida en cada paso. Adicionalmente, se muestra un análisis práctico de la seguridad de la transmisión de los datos.

5.1.1 Ejemplo de ejecución del sistema

Primeramente, debe activarse el sistema del *cloud* que permite obtener los mensajes del *edge* y *fog*, desencriptarlos, verificarlos acorde al *Blockchain* y almacenarlos en la base de datos no relacional. Para ello, se debe ejecutar el programa reader.go en la terminal del sistema operativo Linux en la máquina virtual. Esto es,

```
$ go run reader.go
```

Los mensajes guardados en el bloque son desencriptados y mostrados por pantalla. Por ejemplo, véase la Figura 5-1. Finalmente, en caso de ser válido el bloque, se inserta en la base de datos, tal y como se aprecia en la Figura 5-2.

```
ObjectID("630f5996f6348a05ac03e28a")
(main.Block) {
 Index: (int) 4,
 Time_Stamp: (string) (len=57) "2022-08-31 14:52:30.443429684 +0200 CEST m=+136.775518562",
 MSG: ([]string) (len=11 cap=13) {
  (string) (len=20) "lIdGvw4Bkp2HybQIHELv",
  (string) (len=20) "ZYRt7BuztNoJWUFRGxZY",
  (string) (len=20) "uX4WDEJAh2ieuULY3X8c",
  (string) (len=20) "oRmkCwD3fJhFs0zr5OuA",
  (string) (len=20) "4HZXgaVI0e7sKQQx0ZHv",
  (string) (len=20) "Wk2IJDfjqefpdmvYEOll",
  (string) (len=20) "0Aop1yV3qxSvE23E6fLQ",
  (string) (len=20) "QsATKqZsFyejj2zS65fu",
  (string) (len=20) "5Je46zokbt7y2aYW7eSq",
  (string) (len=20) "tv7E0FiQS3RzxaOulXMb",
  (string) (len=20) "T0CIKTUYHVXPAHZfx5Aq"
 },
 Hash: (string) (len=64) "eea6b3e1b047d0c8fe2bc400ea438fc15e15a5536fc811ece4e77a061bc05943",
 Prev_Hash: (string) (len=64) "62e6c4302afa569412ea11752bf774fa98e75f1630c1acd899f74a264e1a9b4b",
 Statistic: (main.Statistics) {
  RAM_Rasp: (string) (len=4) "9 kB",
  Time_Rasp: (string) (len=10) "736.178µs",
  Size_MSG: (string) (len=9) "500 bytes",
  Size_MSG_Cipher: (string) (len=9) "808 bytes",
  RAM_MV: (string) (len=4) "8 kB",
  Time_MV: (string) (len=9) "69.001µs"
 }
}
```

Figura 5-1 Bloque mostrado por consola en la MV

El contenido del bloque es insertado en la BD una vez comprobado su validez. Véase la Figura 5-2.

Figura 5-2 Contenido del bloque en la BD

A continuación, habría que lanzar el sistema que permite recopilar los mensajes (ya sea desde el ESP32 o desde la propia Raspberry), encriptarlos, encadenarlos en *Blockchain* y enviarlos al *cloud*. Para ello, se debe ejecutar el programa main.go en el terminal de la Raspberry, donde el parámetro proporcionado indica si se opta por la solución 1 o 2. Esto es

$ go run main.go 1 (si se quiere ejecutar la solución 1)
$ go run main.go 2 (si se quiere ejecutar la solución 2)

En la Figura 5-3, se muestra uno de los bloques, ya encadenados al *Blockchain*, que son enviados al *cloud*. En este bloque se observa el mensaje encriptado mediante AES (campo MSG), así como las distintas estadísticas calculadas en la Raspberry, a falta de completar las estadísticas de la máquina virtual. Nótese que por ejemplo, la clave usada para cifrar la información, fichero .k.txt, podría ser la mostrada en la Figura 5-4.

Figura 5-3 Bloque mostrado por consola en la Raspberry Pi

Figura 5-4 Key en el archivo .k.txt

En el caso de que se optara por la solución 1 (sin *edge*) y antes de ejecutar el programa main.go en la Raspberry, habría que lanzar el programa generador de cadenas aleatorias en generateString.py. Para ello, debe abrirse un terminal en la Raspberry y ejecutar:

$ python generateStrings.py

Una posible salida por consola se muestra en la Figura 5-5.

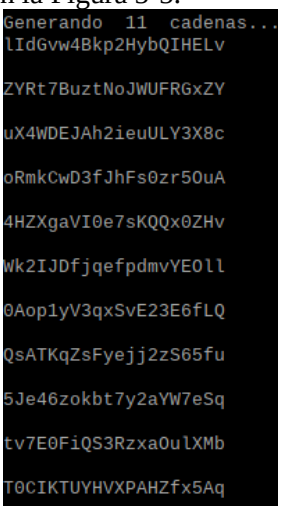

Figura 5-5 Cadenas aleatorias mostradas por consola

Finalmente, si se opta por la solución 2, habría que conectar la ESP32 a un ordenador, cargar el programa ESP32_WIFI.ino en el EPS32. Por último, abrir un terminal en el Arduino IDE para visualizar la ejecución.

Una vez iniciada la ejecución hay dos escenarios posibles:

1. El ESP32 se conectó con el servidor en el *fog*. Entonces, envía un conjunto aleatorio (1-20) de cadenas alfanuméricas aleatorias de 20 caracteres cada una al servidor en el *fog*. Dichas cadenas son mostradas en la consola del Arduino IDE. Véase la Figura 5-6 para un envío de 4 cadenas aleatorias.

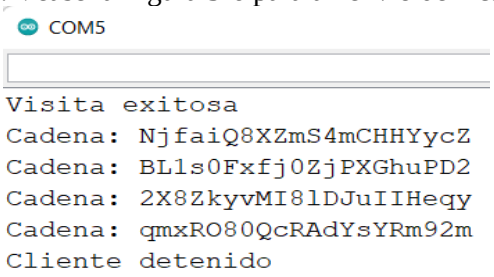

Figura 5-6 Cadenas creadas por el ESP32

2. El ESP32 se conectó a la red, pero no al servidor. En tal caso, en la consola del Arduino IDE, se muestra el mensaje “Acceso fallido” y finaliza el programa. Véase la Figura 5-7.

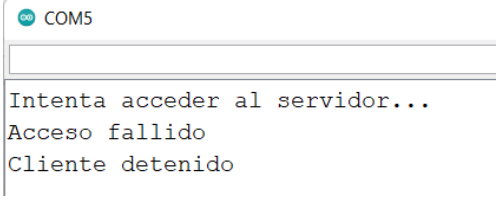

Figura 5-7 ESP32 no consigue conectarse al servidor

En ambos escenarios, tras una espera de 20 segundos se vuelve a tratar de llevar a cabo el envío de cadenas aleatorias de forma indefinida hacia el servidor del *fog*. Nótese que si la conexión a la red falla, entonces se llevará a cabo un proceso automático de reintento de conexión. Véase por ejemplo la Figura 5-8.

El código numérico corresponde a:

- 1: finish scanning AP (Access Point)
- 2: station start
- 201: No AP found

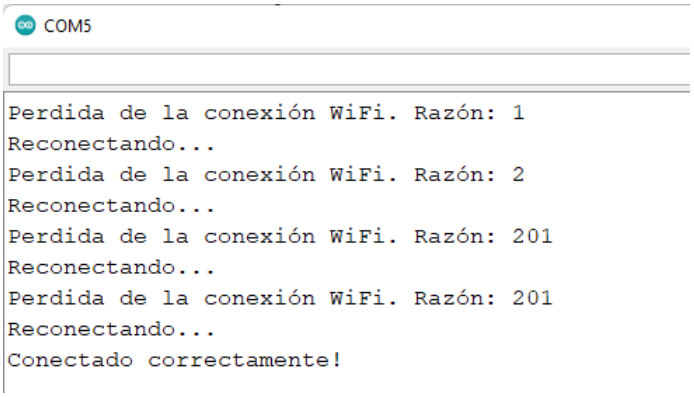

Figura 5-8 Pérdida y restablecimiento de la conexión en el ESP32

5.1.2 Análisis práctico de seguridad entre edge-fog y cloud

El canal SSH proporciona un canal seguro entre el cliente y el servidor. Toda la información viaja cifrada. Si un hacker realiza un ataque, suponiendo que consigue romper el protocolo (algo que hasta ahora no se ha conseguido) e intercepta el mensaje del bloque, aún la información estaría en buen recaudo ya que está encriptada por el cifrado simétrico AES (específicamente el campo MSG del bloque). Sin una clave, no podría descifrar el mensaje y obtener su contenido.

La siguiente imagen muestra el contenido del archivo newBlock.json con el bloque número 7 del *Blockchain*. Dicho JSON fue enviado por SSH de la Raspberry Pi a la MV. Véase la Figura 5-9.

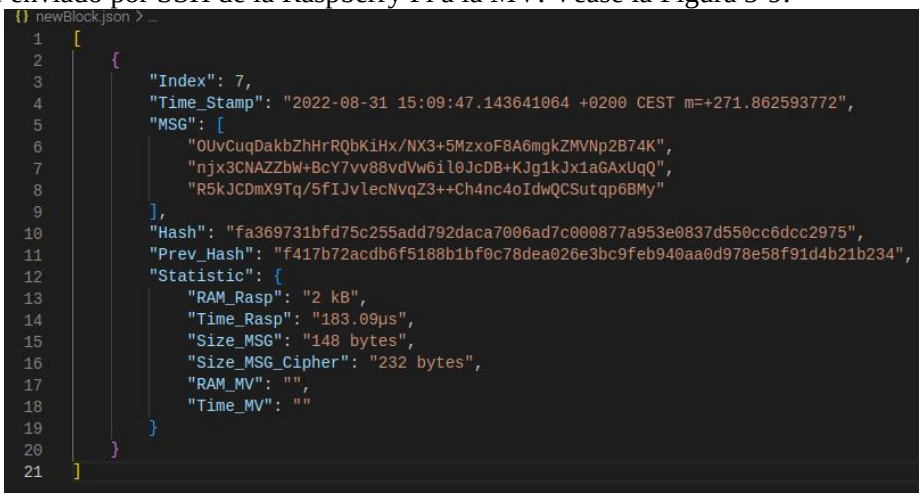

Figura 5-9 Contenido inicial del archivo newBlock.json

En este experimento, se quiere determinar qué pasaría con la cadena de bloques si el archivo newBlock.json, en el trayecto de la Raspberry Pi a la MV por Internet, sufría alguna modificación en sus datos (sin entrar a analizar las posibles causas).

Esta situación fue simulada de la siguiente manera: El archivo es copiado en la carpeta de la máquina virtual */home/linux-vm*, en ese instante y justo antes de que fuera leído por el programa reader.go, se cambió un carácter en el bloque, específicamente en el campo MSG. La alteración fue simple, una ‘O’ por una ‘u’, como se puede ver en la Figura 5-10.

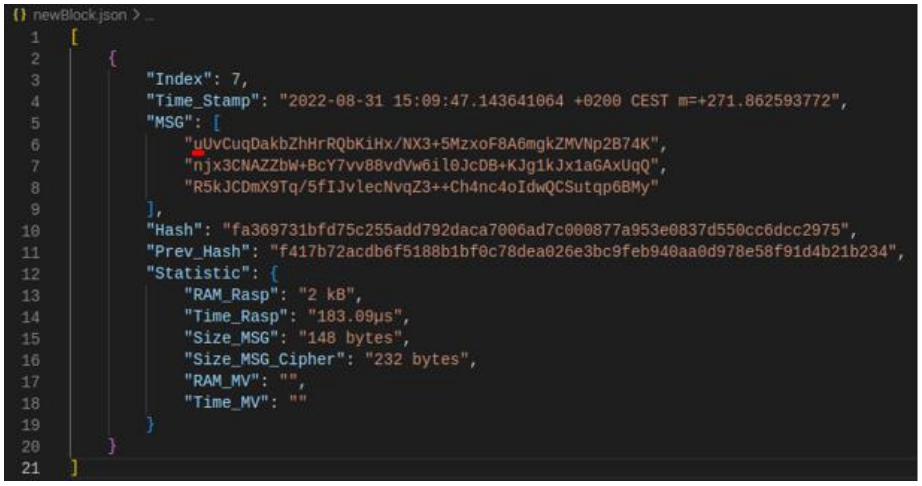

Figura 5-10 Contenido modificado del archivo newBlock.json

Tal y como se detalló previamente, la función isBlockValid() valida el bloque antes de introducirlo en la base de datos. Para ello, comprueba si su Index, Hash y Prev_Hash son los esperados. Así, al calcular el valor del Hash para el bloque y compararlo con el valor proporcionado en el propio bloque, es posible detectar rápidamente que hubo una modificación o corrupción en la información del mensaje. Consecuentemente, el resultado del experimento fue el esperado. Salta el error pertinente, el programa finaliza y el bloque no es insertado en la BD. Véase la Figura 5.11.

```
Calculated:  1fcd762608de92ceb620630d79ce6458ade7ea36f9f4d677d20cd7e29a9cc888
Stored:  fa369731bfd75c255add792daca7006ad7c000877a953e0837d550cc6dcc2975
CalculateHash problem
panic: Bloque incorrecto

goroutine 1 [running]:
main.main()
        /home/linux-vm/TFG/reader.go:80 +0x36d
exit status 2
linux-vm@Linux-VM:~/TFG$
```

Figura 5-11 Error en la validación del bloque

5.2 Análisis experimental

Para obtener los datos experimentales se modificó el programa generateStrings.py para que el número de cadenas generadas para cada nuevo bloque fuera una secuencia ascendente (empezando en 1 cadena y terminado en 20 cadenas), eliminando así el factor aleatorio visto en la solución 1. Se realizaron dos pruebas: la primera consistía en obtener valores estadísticos con cadenas de longitud fija igual a 20 caracteres (véase la Tabla 2) y la segunda prueba con cadenas de longitud fija igual a 2000 caracteres (véase la Tabla 3). El tiempo total de cada prueba fue de aproximadamente 400 segundos, ya que el sistema crea un bloque cada 20 segundos.

Tanto en la Tabla 2 como en la 3, se muestra la cantidad de caracteres enviados en el bloque, junto a una serie de variables estadísticas calculadas durante el proceso de cifrado y descifrado. Esto es, para la Raspberry, RAM del cifrado, tamaño sin cifrar, tamaño cifrado y tiempo requerido para cifrar. En la máquina virtual, por su parte, las variables tomadas fueron el tiempo para desencriptar y la RAM consumida en el proceso.

Cantidad de caracteres	RAM del cifrado (kB)	Tiempo de cifrado (ms)	Tamaño sin cifrar (bytes)	Tamaño del cifrado (bytes)	RAM del descifrado (kB)	Tiempo de descifrado (ms)
20	0	413,345	60	88	1	91,3
40	1	637,508	96	152	1	124,099
60	3	221,682	148	232	2	60,8
80	3	642,507	168	280	3	122,799
100	4	431,938	252	392	4	53,799

120	5	686,175	272	440	5	119,499
140	6	604,195	292	488	5	78,8
160	6	664,193	312	536	6	119,6
180	7	540,936	460	712	8	99,4
200	8	628,527	480	760	8	57,7
220	9	561,954	500	808	8	54,8
240	9	814,581	520	856	9	169,999
260	10	895,729	540	904	11	72,699
280	11	747,045	560	952	11	142,3
300	12	656,101	580	1000	12	155,4
320	13	758,378	600	1048	12	64,5
340	13	978,021	876	1352	14	140
360	14	880,152	896	1400	14	63,8
380	15	842,838	916	1448	15	144,698
400	16	324,013	936	1496	16	117,699

Tabla 2 Datos estadísticos del cifrado y descifrado de cadenas de tamaño 20 caracteres

A simple vista, en la Tabla 2 se observa que la RAM del cifrado y el descifrado de las cadenas tienen valores muy parecidos, siendo estos relativamente pequeños y asumibles para dispositivos del *edge*/*fog*. También se observa que se consume más tiempo en el cifrado que en el descifrado, lo cual es de esperar debido a las diferencias de potencia de cómputo, no siendo valores excesivos en el caso del *edge*/*fog*. Por último, el tamaño de la variable sin cifrar es menor que la cifrada, pero no implica ninguna problemática relevante.

Nos preguntamos si esta tendencia seguiría cuando se aumentara la longitud de las cadenas, por lo que se decidió aumentar 1000 veces la longitud de las cadenas. Los datos obtenidos fueron recopilados en la Tabla 3.

Cantidad de caracteres	RAM del cifrado (kB)	Tiempo de cifrado (ms)	Tamaño sin cifrar (bytes)	Tamaño del cifrado (bytes)	RAM del descifrado (kB)	Tiempo de descifrado (ms)
2000	12	289,647	2040	2728	8	87,299
4000	24	87,764	4056	5432	15	118,698
6000	35	1145,157	6088	8152	22	107,899
8000	48	1517,987	8088	10840	29	122,799
10000	59	1638,893	10152	13592	37	139,498
12000	72	1413,988	12152	16280	44	117,999
14000	83	908,992	14152	18968	51	141,998
16000	95	2466,775	16152	21656	58	153,799
18000	107	2452,368	18280	24472	66	166,798
20000	119	923,788	20280	27160	73	159,698
22000	131	2349,202	22280	29848	80	198,698
24000	142	1031,528	24280	32536	88	193,898
26000	154	1114,898	26280	35224	95	204,198
28000	166	1191,787	28280	37912	102	165,998
30000	178	1403,192	30280	40600	110	178,198
32000	190	1267,434	32280	43288	117	192,198
34000	202	1371,025	34536	46232	124	194,698
36000	214	1456,728	36536	48920	131	200,298
38000	226	1574,579	38536	51608	138	210,698
40000	237	1537,616	40536	54296	146	222,397

Tabla 3 Datos estadísticos del cifrado y descifrado de cadenas de tamaño 2000 caracteres

Analizando con mayor detenimiento el tiempo de cifrado y descifrado, en la Figura 5-12 el tiempo de cifrado es bastante mayor que el de descifrado, para cadenas de 20 caracteres. Se observa además que el tiempo de descifrado es más constante con respecto a la cantidad de caracteres.

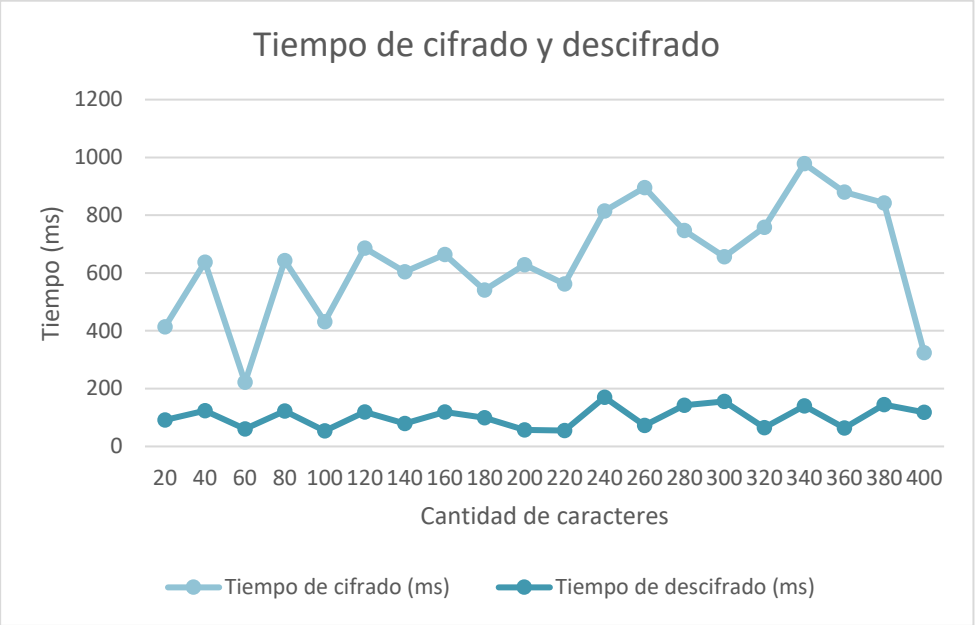

Figura 5-12 Gráfica del tiempo de cifrado y descifrado de cadenas de 20 caracteres

Como se puede observar en la Figura 5-13, los valores del tiempo de descifrado con cadenas de longitud 2000 caracteres son muy parecidos a los tiempos del descifrado con cadenas de 20 caracteres. En cambio, hay un aumento significativo en el tiempo de cifrado con respecto al descifrado como se puede evidenciar en la Tabla 3. Se observa por tanto que el dispositivo del *edge/fog* es más sensible al incremento de la cantidad de caracteres para el proceso de cifrado.

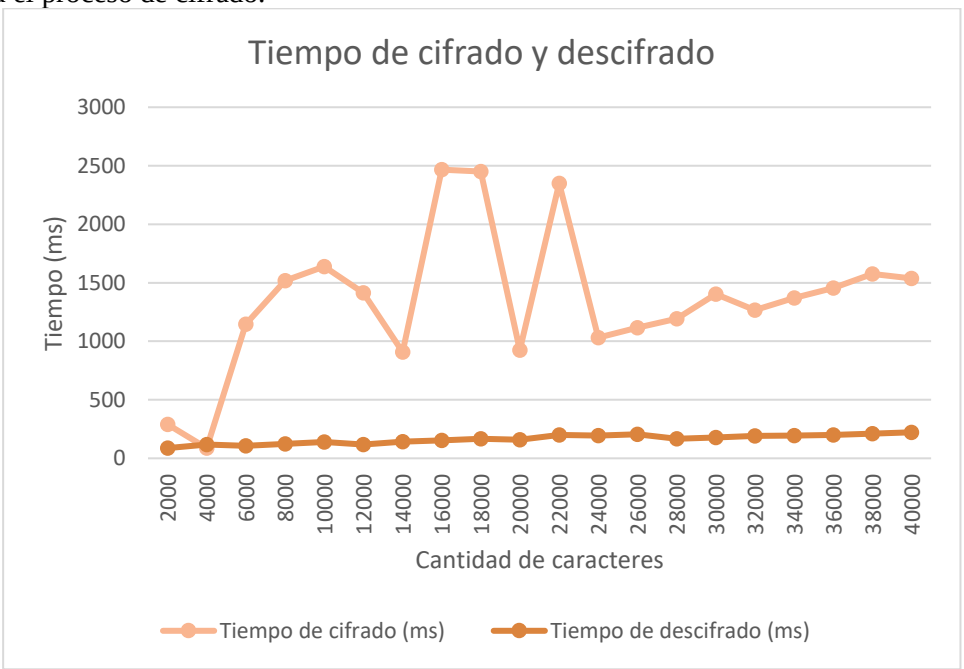

Figura 5-13 Gráfica del tiempo de cifrado y descifrado de cadenas de 2000 caracteres

Conclusión: El tiempo del cifrado, en el *edge/fog*, es muy variable y mayor con respecto al tiempo de descifrado a medida que aumentan el número de cadenas.
Como se puede evidenciar en la Tabla 2, el tamaño de las cadenas que no están cifradas varía de 60 a 936 bytes y de 88 a 1496 bytes en cadenas cifradas. El tamaño de las cadenas cifradas es considerablemente mayor

que las no cifradas. Véase la Figura 5-14.

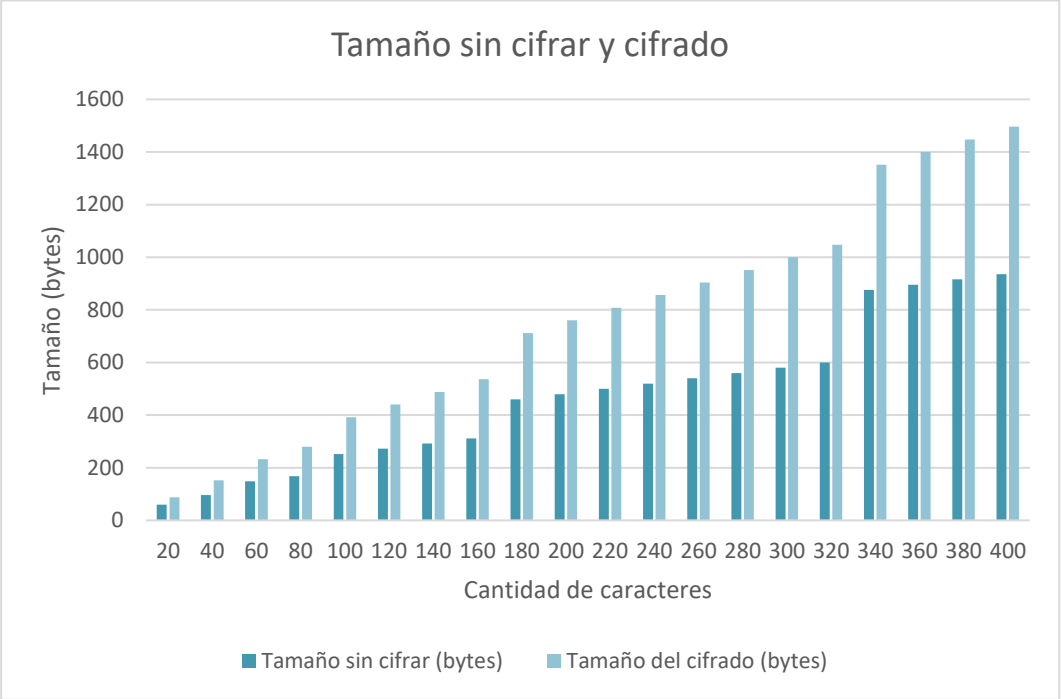

Figura 5-14 Gráfica del tamaño sin cifrar y cifrado de cadenas de 20 caracteres

La diferencia entre el tamaño del cifrado y el no cifrado es mayor a medida que la cantidad de caracteres aumenta. Por ejemplo, como se vio en la Tabla 3, para 40000 caracteres el tamaño del no cifrado es de 40536 y 53296 bytes para el cifrado. Véase la Figura 5-15.

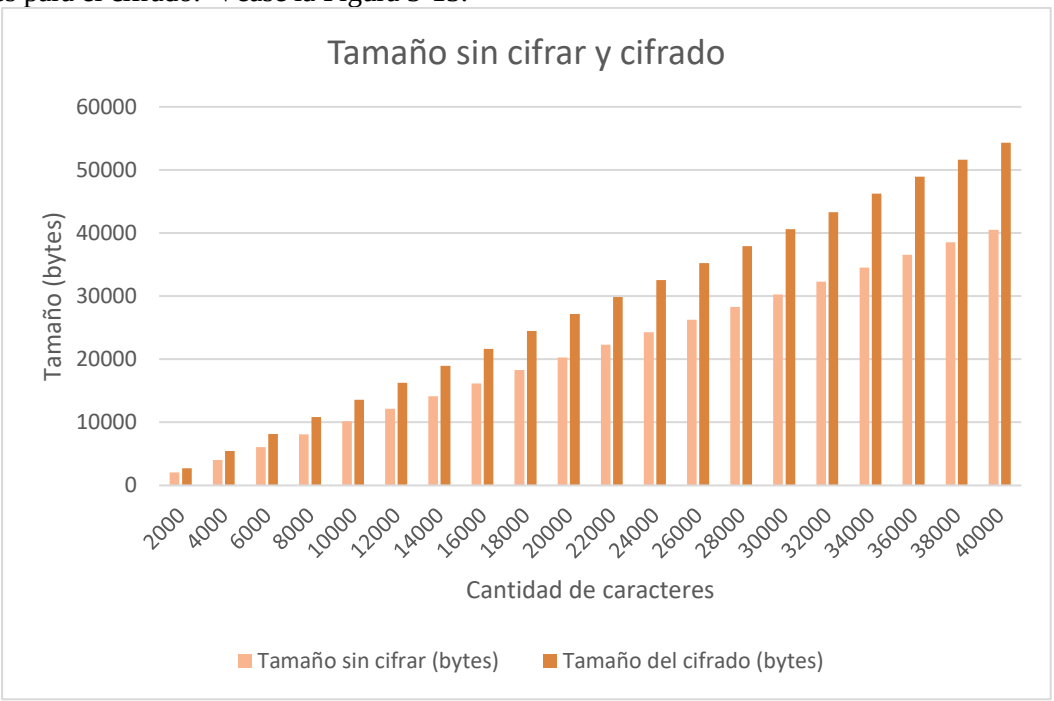

Figura 5-15 Gráfica del tiempo de cifrado y descifrado de cadenas de 2000 caracteres

Conclusión: El tamaño de la variable sin cifrar es mayor que la variable cifrada a medida que aumentan la cantidad de cadenas.

Como se puede observar en la Tabla 2, la RAM del cifrado y descifrado para las cadenas de longitud 20 caracteres hay un crecimiento casi constante de estas variables, empezando en 0 KB y terminado en 16 KB. Véase la Figura 5-16.

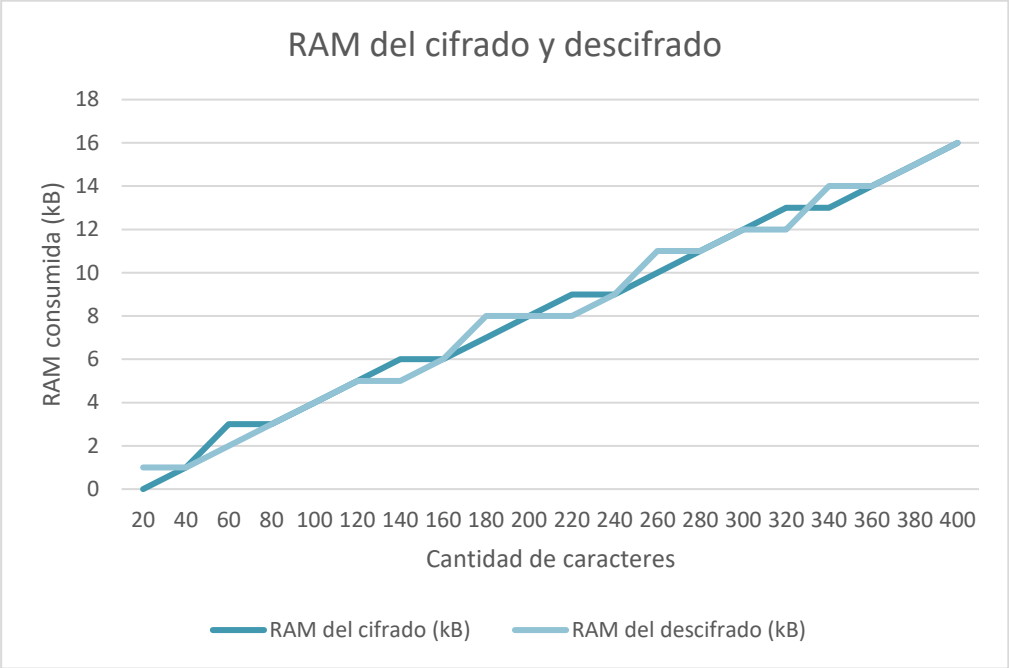

Figura 5-16 Gráfica de la RAM del cifrado y descifrado de cadenas de 20 caracteres

El crecimiento de RAM del cifrado es considerablemente mayor que la RAM del descifrado empezando en 12 KB para 2000 caracteres y terminando en 237 KB con 40000 caracteres como se puede ver en la Tabla 3. Existe un crecimiento lineal en ambas variables como se puede ver en la Figura 5-17.

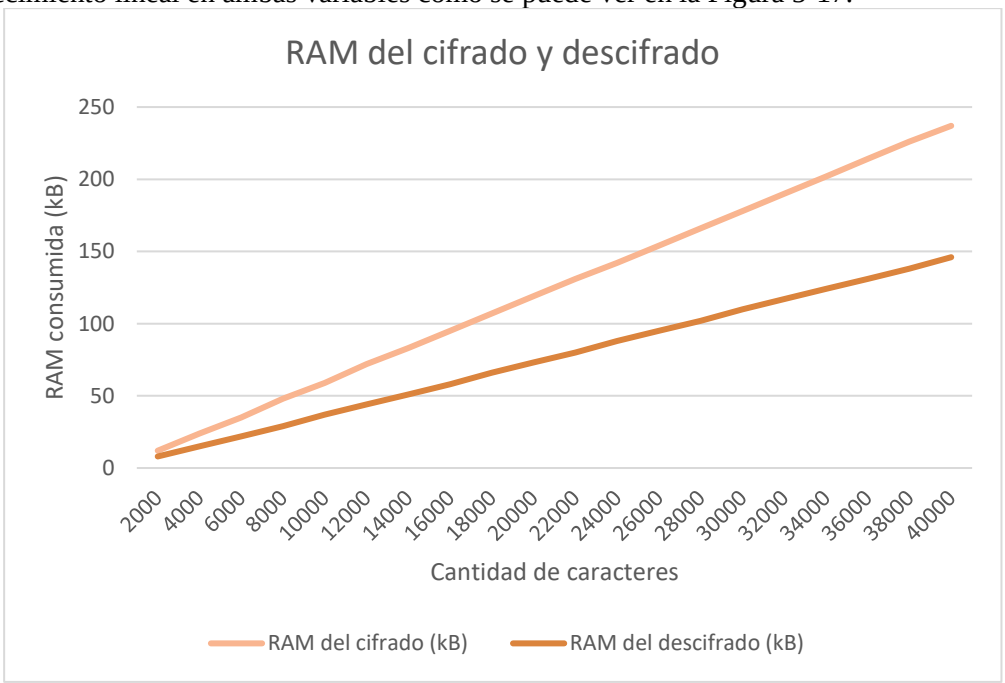

Figura 5-17 Gráfica de la RAM del cifrado y descifrado de cadenas de 2000 caracteres

Conclusión: La variable de RAM del cifrado tenía valores muy similares a la RAM del descifrado cuando las cadenas tenían una longitud de 20 caracteres. Al aumentar a 2000 la longitud, la variable de la RAM del cifrado aumentó considerablemente con respecto a la RAM del descifrado a medida que la cantidad de cadenas aumentaba.

Como conclusión a este estudio, se puede afirmar que es posible llevar a cabo la implementación se la seguridad de las comunicaciones sensibles desde el *edge*/*fog* hasta el *cloud*, mediante *Blockchain* y técnicas de encriptación. Pero, sin lugar a duda, se debe tomar especial atención al tamaño de la información enviada y su periocidad, de modo que el sistema cumpla con los restringidos requisitos de *hardware* (memoria RAM principalmente) y temporización (tiempo de respuesta) en este tipo de aplicaciones.

6 Conclusiones y líneas futuras

En este capítulo se expondrán las principales conclusiones que se ha llegado en realización de este TFG así como algunas de las mejoras que puede recibir el sistema en futuros desarrollos.

6.1 Conclusiones

En cuanto a los objetivos propuestos en el proyecto, se puede afirmar que se ha completado el objetivo principal que era crear una solución que permitiera el envío de datos seguros entre dispositivos IoT, del *edge-fog* al *cloud*, empleado la tecnología *Blockchain*.

Para la caracterización de esta solución, se ha llevado a cabo un estudio experimental donde se han tomado un conjunto de estadísticas a la hora de enviar datos asegurados mediante *Blockchain* entre la Raspberry Pi (*fog*) y la máquina virtual de Azure (*cloud*). Esta información estadística fue el tiempo de procesado, la memoria RAM empleada y el tamaño de las cadenas en el proceso de cifrar y descifrar la información enviada. Durante este estudio se enviaron mensajes con distinta longitud, y aunque esta información se logró enviar asegurada mediante *Blockchain* entre los dispositivos sin mayor problema, es necesario tener en cuenta la longitud de las cadenas ya que los recursos del IoT son limitados, especialmente en el *edge*. Las conclusiones obtenidas durante este estudio fueron las siguientes:

- El tiempo del cifrado es muy variable y mayor con respecto al tiempo de descifrado a medida que aumentan el número de caracteres.
- El tamaño de la variable sin cifrar es mayor que la variable cifrada a medida que aumentan la cantidad de caracteres.
- La cantidad de RAM usada durante el cifrado aumentó considerablemente con respecto a la RAM del descifrado, a medida que la cantidad de caracteres aumentaba.
- Se puede afirmar que es posible llevar a cabo la implementación se la seguridad de las comunicaciones sensibles desde el *edge*/*fog* hasta el *cloud*, mediante *Blockchain* y técnicas de encriptación. Pero, sin lugar a duda, se debe tomar especial atención al tamaño de la información enviada y su periocidad, de modo que el sistema cumpla con los restringidos requisitos de *hardware* y temporización en este tipo de aplicaciones.

En el plano personal, antes de llevar a cabo este proyecto me llamaba la atención la tecnología *Blockchain* y gracias a este desarrollo he podido conocer más a fondo su funcionamiento, tipos y beneficios. He aprendido más sobre el IoT, el cifrado y las bases de datos. El desarrollo de este proyecto me ha conllevado a recordar lenguajes como Arduino y Python, o tener un nuevo aprendizaje al trabajar con el lenguaje Golang o con bases de datos no relacionales, que ha complementado la formación obtenida durante la titulación y ha asentado la base para poder desarrollar futuros proyectos en este ámbito.

6.2 Líneas futuras

El proyecto desarrollado es susceptible de ser mejorado en distintos aspectos, aunque cumple con la idea principal. A continuación, se describen algunas de las ideas para futuros desarrollos:

- El ESP32 se podría sustituir por alguno de los dispositivos reales usados en Bindi, como la pulsera o el colgante, para realizar pruebas reales.
- Los archivos enviados entre la Raspberry Pi y la MV se realizan de forma segura gracias al protocolo SSH. En cambio, las cadenas enviadas del ESP32 a la Raspberry Pi no viajan encriptadas. Una solución podría llegar a ser implementar en Arduino un certificado TLS (es el acrónimo de Transport Layer Security). TLS es un protocolo de cifrado de la capa de transporte de Internet y su función principal es cifrar el tráfico entre un cliente y un servidor.

- Para realizar las pruebas se ha instalado en local, en la máquina virtual de Azure, *MongoDB*. Esta versión es gratis, no tiene problemas de almacenamiento porque emplea el de la máquina virtual, pero tiene una gran desventaja y es que solo acepta conexiones del propio ordenador. A futuro, se podría instalar *MongoDB* Atlas es una plataforma de datos para aplicaciones multicloud. Esto permitiría administrar, escalar y automatizar la base de datos de una forma segura y fiable, aceptando únicamente las conexiones de las IPs autorizadas. Aunque no es gratis, tiene un gran abanico de posibilidad de acuerdo con las necesidades de los clientes que solicitan sus servicios.
- Cada vez que se ejecuta el programa main.go, visto en capítulos anteriores, se crea una nueva cadena de bloques. Si se quiere seguir introduciendo bloques a una cadena creada en una ejecución previa, habría que conectar el último bloque introducido en la base de datos con el bloque nuevo creado.
- Se podría mejorar el cifrado de la información en el bloque mediante el uso de un cifrado híbrido. En este tipo de cifrado se incorpora una combinación de cifrado asimétrico y simétrico para aprovechar las ventajas de cada forma de cifrado.
- En la implementación realizada en la máquina virtual, véase la Sección 5.1.2, si un bloque no es válido éste no se inserta a la cadena de bloque y el programa del *cloud* se cierra. Esto implica que si se siguieran enviando nuevos bloques, no sería leídos ni insertados a la base de datos. Una posible opción sería notificarle al programa main.go que la información del bloque no válido la debe volver a enviar a la máquina virtual y los nuevos bloques tienen que ir conectado con el último bloque válido.

Bibliografía

[1] A. CM, «Qué es Blockchain: Significado y Definición de esta tecnología,» [En línea]. Available: https://aulacm.com/que-es/blockchain-significado-definicion-tecnologia/.

[2] P. B. O. &. B. S. Gokhale, «Introduction to IOT,» *International Advanced Research Journal in Science, Engineering and Technology,* pp. 41-44, 2018.

[3] D. Abogados, «Delito de violencia de género,» Dexia Abogados, 20 diciembre 2021. [En línea]. Available: https://www.dexiaabogados.com/blog/violencia-de-genero/.

[4] M. Redondo, «El doble filo de las pulseras telemáticas para las víctimas de violencia de género,» Hipertextual, 24 noviembre 2021. [En línea]. Available: https://hipertextual.com/2021/11/pulseras-telematicas-violencia-genero.

[5] U. C. I. d. Madrid, «EMPATIA,» Portal Universidad Carlos III de Madrid, [En línea]. Available: https://portal.uc3m.es/portal/page/portal/inst_estudios_genero/proyectos/EMPATIA.

[6] E. R.-G. C. L.-M. M. F. C. A. R. B. J. M. L.-G. C. P.-M. C. L.-O. Jose A. Miranda, «Bindi: Affective Internet of Things to Combat,» *JOURNAL OF LATEX CLASS FILES,* vol. 14, nº 8, 2015.

[7] IBM, «¿Qué es la tecnología de blockchain?,» [En línea]. Available: https://www.ibm.com/es-es/topics/what-is-blockchain.

[8] M. A. L. -. V. C. Unda, «Aprende los tres elementos clave de blockchain con este ejemplo práctico,» 28 June 2018. [En línea]. Available: https://blogs.iadb.org/conocimiento-abierto/es/elementos-clave-de-blockchain/.

[9] IBM, «¿Qué es la tecnología de blockchain?,» [En línea]. Available: https://www.ibm.com/es-es/topics/what-is-blockchain.

[10] J. Castelan, «¿Qué es el blockchain y cómo funciona? 5 beneficios de esta tecnología,» 30 junio 2022. [En línea]. Available: https://talently.tech/blog/que-es-blockchain/.

[11] bsm, «Tipos de blockchain: pública, privada e híbrida,» 15 octubre 2021. [En línea]. Available: https://www.bsmexecutive.com/diferencias-entre-blockchain-publica-privada-e-hibrida/.

[12] M. García, «¿Qué es IoT (Internet Of Things)? | Deloitte Spain,» [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html.

[13] G. M. J.-S. L. T. R. Jorge Portilla, «The Extreme Edge at the Bottom of the Internet of Things: A Review,» *IEEE Sensors Journal,* vol. 19, nº 9, pp. 3179-3190, 2019.

[14] V. C. A. Y. D. N. B. a. J. O. R. Morabito, «Consolidate IoT Edge Computing with Lightweight Virtualization,» *IEEE Network,* vol. 32, nº 1, pp. 102-111, 2018.

[15] Enertic, «¿Qué es el IoT Edge?,» 19 marzo 2019. [En línea]. Available: https://enertic.org/que-es-el-iot-edge/.

[16] M. C. y. T. Zhang, «Fog and IoT: And Overview of Research Opportunities,» *IEEE Internet of Things Journal,* vol. 3, nº 6, pp. 854-864, diciembre de 2016.

[17] IBM, «¿Qué es cloud computing?,» [En línea]. Available: https://www.ibm.com/es-es/cloud/learn/cloud-computing-gbl.

[18] J. Chen, «Azure Fundamental: IaaS, PaaS, SaaS | Medium,» 19 7 2020. [En línea]. Available: https://medium.com/chenjd-xyz/azure-fundamental-iaas-paas-saas-973e0c406de7.

[19] R. Hat, «Tipos de cloud computing,» 15 marzo 2018. [En línea]. Available: https://www.redhat.com/es/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-cloud.

[20] Kaspersky, «¿Qué es el cifrado de datos? Definición y explicación,» [En línea]. Available: https://latam.kaspersky.com/resource-center/definitions/encryption.

[21] S. Eléctronica, « ¿Qué es la Encriptación o Cifrado?,» [En línea]. Available: https://www.sede.fnmt.gob.es/preguntas-frecuentes/otras-preguntas/-/asset_publisher/1RphW9IeUoAH/content/1024-que-es-la-encriptacion-o-cifrado-?inheritRedirect=false.

[22] MDN, «Criptografía de clave simétrica - Glosario,» 15 agosto 2022. [En línea]. Available: https://developer.mozilla.org/es/docs/Glossary/Symmetric-key_cryptography.

[23] C. FNMT, «El Cifrado Digital,» [En línea]. Available: https://www.cert.fnmt.es/curso-de-criptografia/criptografia-de-clave-simetrica.

[24] I. D. Guide, «Cifrado asimétrico,» [En línea]. Available: https://www.ionos.es/digitalguide/servidores/seguridad/cifrado-asimetrico/.

[25] A. López, «Todo sobre criptografía: Algoritmos de clave simétrica y asimétrica,» 9 agosto 2022. [En línea]. Available: https://www.redeszone.net/tutoriales/seguridad/criptografia-algoritmos-clave-simetrica-asimetrica/.

[26] Keepcoding, «¿Qué es el algoritmo AES?,» 25 julio 2022. [En línea]. Available: https://keepcoding.io/blog/que-es-el-algoritmo-aes/.

[27] boxcryptor, «Cifrado AES y RSA,» [En línea]. Available: https://www.boxcryptor.com/es/encryption/.

[28] Ceupe, «¿Qué es una Base de datos? Características, componentes y tipos,» 6 2 2022. [En línea]. Available: https://www.ceupe.com/blog/base-de-datos.html?dt=1661997390161.

[29] Oracle, «¿Qué es una base de datos relacional?,» [En línea]. Available: https://www.oracle.com/es/database/what-is-a-relational-database/.

[30] F. Tablado, «Base de datos no relacional. ¿Qué es? Características y ejemplos,» 9 9 2020. [En línea]. Available: https://ayudaleyprotecciondatos.es/bases-de-datos/no-relacional/.

[31] U. Electronics, «ESP32 38 Pines ESP WROOM 32,» [En línea]. Available: https://uelectronics.com/producto/esp32-38-pines-esp-wroom-32/.

[32] Espressif, «Serie ESP32,» [En línea]. Available: https://www.espressif.com/en/products/socs.

[33] R. Pi, «Raspberry Pi 4 Model B specifications,» [En línea]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[34] K. t. Smart, «Raspberry Pi 4 GPIO Pinout,» 6 Noviembre 2019. [En línea]. Available: https://keytosmart.com/single-board-computers/raspberry-pi-4-gpio-pinout/.

[35] DBandTech.com, «MySQL vs MongoDB - DBandTech.com,» [En línea]. Available: https://dbandtech.com/mysql-vs-mongodb/.

[36] D. Taylor, «MongoDB vs MySQL - Diferencia entre ellos,» 29 agosto 2022. [En línea]. Available: https://www.guru99.com/mongodb-vs-mysql.html.

[37] C. Health, «Part 2: Networking — Code your own blockchain in less than 200 lines of Go!,» 6 2 2018. [En línea]. Available: https://mycoralhealth.medium.com/part-2-networking-code-your-own-blockchain-in-less-than-200-lines-of-go-17fe1dad46e1.

[38] M. Azure, «Cree soluciones en la nube con una cuenta gratuita de Azure,» [En línea]. Available: https://azure.microsoft.com/es-es/free/.

[39] Y. Zhang, «Blockchain. In Encyclopedia of Wireless Networks,» *Springer International Publishing,* pp. 115-118, 2020.

[40] D. Purón, «IoT Edge Computing, Nodos Edge y casos de uso en el sector Industrial,» 17 noviembre 2021. [En línea]. Available: https://barbaraiot.com/blog/iot-edge-computing-nodos-edge/.

[41] ciberseg1922, «Capas de IoT que debes conocer | Ciberseguridad,» 8 11 2021. [En línea]. Available: https://ciberseguridad.com/guias/nuevas-tecnologias/capas-iot/.

[42] Azure, «Developer tools, technical documentation and coding examples | Microsoft Docs,» 23 6 2022. [En línea]. Available: https://docs.microsoft.com/es-es/azure/cloud-adoption-framework/get-started/what-is-azure.

[43] Azure, «Cloud Computing Services | Microsoft Azure,» [En línea]. Available: https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-a-virtual-machine/#how-do-work.

[44] Oracle, «Modelo de arquitectura del protocolo TCP/IP (Guia de administracion del sistema: servicios IP),» [En línea]. Available: https://docs.oracle.com/cd/E19957-01/820-2981/ipov-10/.

[45] L. Llamas, «Configurar IP estática en Raspberry Pi,» 27 mayo 2018. [En línea]. Available: https://www.luisllamas.es/raspberry-pi-ip-estatica/.

[46] A. CM, «Qué es Blockchain: Significado y Definición de esta tecnología,» [En línea]. Available: https://aulacm.com/que-es/blockchain-significado-definicion-tecnologia/.

[47] U. Electronics, «ESP32 38 Pines ESP WROOM 32,» [En línea]. Available: https://uelectronics.com/producto/esp32-38-pines-esp-wroom-32/.

[48] P. Herreros, «Blockchain: Qué es y cómo funciona (con ejemplos fáciles),» [En línea]. Available: https://pabloherreros.com/que-es-blockchain/.

[49] D. Bitcoin, «Blockchain,» [En línea]. Available: https://www.diariobitcoin.com/glossary/blockchain-2/.

[50] SAP, «What is blockchain technology?,» [En línea]. Available: https://www.sap.com/insights/what-is-blockchain.html.

Apéndice A
Anexo I: Presupuesto

En este apartado, se especifica el presupuesto para el desarrollo del proyecto proporcionando el coste de material empleado y los recursos humanos necesarios. Dichos costes se dividen en: coste de equipamiento, software, recursos humanos y coste total.

A.1. Coste de equipamiento

El coste del equipamiento se divide en dos partes. Por un lado, el coste de los componentes del hardware (véase la Tabla 4), que incluye los dispositivos IoT y todo el material necesario para su uso (cables, adaptadores, monitores…). Por otro lado, el coste del equipo de desarrollo utilizado, véase la Tabla 5. Para el cálculo de coste del equipo de desarrollo se ha estimado que la vida útil del portátil utilizado es de cuatro años, mientras que la duración del proyecto es de cuatro meses. Así, el coste total del equipamiento se encuentra en la Tabla 6.

Componente	Precio
1x ESP32	16,34 €
1x Cable micro-USB a USB	1,50 €
1x Raspberry PI 4 Modelo B	179,00 €
1x Cargador Tipo C	11,99 €
1x Tarjeta micro USB de 64 GB	13,90 €
1x Lector de Tarjetas SD/Micro SD	15,99 €
1x Cable micro HDMI a HDMI	7,49 €
1x Monitor HP	109,00 €
1x Teclado	58,99 €
1x Ratón	19,99 €
Desglose del coste total HW	**439,19 €**

Tabla 4 Coste de hardware

Componente	Precio
Ordenador portátil Lenovo Legion Y530-15ICH Intel Core i7-8750H CPU @ 2.20GHz 2.21GHz Memoria RAM DDR4 16GB Disco duro SSD 250 GB Tarjeta gráfica GeForce GTX 1050	900 €
Coste imputable (4 meses)	**75 €**

Tabla 5 Coste del equipo utilizado para el desarrollo

Componente	total
Coste de hardware	439,19 €
Coste del equipo de desarrollo	75 €
Desglose del coste total del equipamiento	**509,19 €**

Tabla 6 Coste del equipamiento requerido

A.2. Coste de licencias software

En este capítulo se recoge el presupuesto requerido para adquirir las herramientas software requeridas para el desarrollo, véase Tabla 7. La mayoría del software utilizado es de libre acceso, por lo que no incurre coste. En el caso de esta máquina virtual, su precio de pago por uso es de 81,00€/mes, requiriendo su uso durante cuatro meses.

Componente	Licencia	Precio
Microsoft Windows 10	EULA	145,00 €
Git	GPL	0 €
Visual Studio Code	MIT	0 €
Arduino IDE	GNU	0 €
Máquina Virtual Azure	Pago por uso	324,00 €[27]
Desglose del coste de software		**469,00 €**

Tabla 7 Coste de las licencias software requeridas

A.3. Coste de recursos humanos

El estudiante y el tutor fueron los únicos que participaron en el desarrollo del proyecto. Sin embargo, en este capítulo se ha querido mostrar el coste incurrido por cada una de las figuras habituales en un proyecto de desarrollo, según las horas requeridas por el estudiante y el tutor.
En la Tabla 8 se encuentra detallado este coste. Las figuras involucradas tienen las siguientes responsabilidades:

- Jefe de proyecto. Encargado de la planificación y monitorización del sistema, además de la toma de las decisiones para la consecución de los objetivos del proyecto.
- Analista programador. Encargado del análisis y diseño de los módulos del sistema, además del análisis de las tecnologías empleadas.
- Programador. Desarrollo de los módulos del sistema, la depuración del código y pruebas sobre el sistema.
- Secretaría. Incluye la redacción de la memoria y de los documentos adicionales sobre el desarrollo del sistema.

Concepto	Horas	Coste/horas	Coste total
Jefe de proyecto	10	70	700 €
Analista Programador	35	44	1540 €
Programador	200	32	6400 €
Secretaría	100	24	2400 €
Total			**11040 €**

Tabla 8 Desglose del coste de recursos humanos

[27] Resultado de 4 meses de trabajo a razón de 80€/mes

A.4. Coste total

En la Tabla 9 se detalla el coste total del proyecto como suma del coste de equipamiento, de licencias software y de recursos humanos. Además, se incluye un apartado de costes generales relacionados con gastos de oficina y mantenimiento de las instalaciones que supone el 15% del coste de equipamiento, licencias de software y recursos humanos.

Componente	**Precio**
Coste de equipamiento	509,19 €
Costes de licencias software	469,00 €
Costes de recursos humanos	11040,00 €
Costes generales	1802,73 €
Coste total del proyecto	**13820,92 €**

Tabla 9 Desglose del coste total del proyecto

El coste total del proyecto asciende a la cantidad de TRECE MIL OCHOCIENTOS VEINTE EUROS CON NOVENTA Y DOS CÉNTIMOS.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá