UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

IMPLEMENTACIÓN DE UN PROTOTIPO DE UN SISTEMA PARA LA INTEGRIDAD DEL VÍDEO EN TIEMPO REAL UTILIZANDO BLOCKCHAIN Y RASPBERRY PI

Salvador García Jiménez

Tutor: Juan Carlos Guerri Cebollada

Cotutor: Pau Arce Vila

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster en Ingeniería Telecomunicación

Curso 2019-20

Valencia, 6 de julio de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Resumen

El avance tecnológico en el ámbito audiovisual ha conseguido que grabaciones de contenido multimedia sean de vital importancia en diferentes ámbitos. Con el fin de garantizar la integridad de estos archivos y que se consideren válidos, este TFM propone un prototipo compuesto de una cámara y una Raspberry Pi. La Raspberry Pi se encargará de procesar el vídeo de la cámara para emitirlo en directo a través de un servidor web. Este vídeo se emitirá en formato de bitrate adaptativo para mejorar la calidad de experiencia del usuario que consume el vídeo. Gracias a un sistema basado en blockchain, una serie de procesos analizarán los segmentos de vídeo generados, almacenando información resumida de su contenido de manera inmutable. De este modo, se tratará de poner a prueba si registrar el contenido audiovisual en tiempo real dentro de la blockchain es factible. A partir de esta solución se pretende evitar que la composición del vídeo sea modificada acortando el tiempo entre la emisión y su registro tanto como sea posible. Mediante este método es posible comprobar, en tiempo real o a posteriori, si un vídeo ha sido alterado desde el momento de la emisión.

Resum

L'avanç tecnològic en l'àmbit audiovisual ha aconseguit que enregistraments de contingut multimèdia siguen de vital importància en diferents àmbits. Amb la finalitat de garantir la integritat d'aquests arxius i que es consideren vàlids, aquest TFM proposa un prototip compost d'una càmera i una Raspberry Pi. La Raspberry Pi s'encarregarà de processar el vídeo de la càmera per a emetre'l en directe a través d'un servidor web. Aquest vídeo s'emetrà en format de bitrate adaptatiu per a millorar la qualitat d'experiència de l'usuari que consumeix el vídeo. Gràcies a un sistema basat en blockchain, una sèrie de processos analitzaran els segments de vídeo generats, emmagatzemant informació resumida del seu contingut de manera immutable. D'aquesta manera, es tractarà de posar a prova si registrar el contingut audiovisual en temps real dins de la blockchain és factible. A partir d'aquesta solució es pretén evitar que la composició del vídeo siga modificada acurtant el temps entre l'emissió i el seu registre tant com siga possible. Mitjançant aquest mètode és possible comprovar, en temps real o a posteriori, si un vídeo ha sigut alterat des del moment de l'emissió.

Abstract

Technological progress in the audiovisual field has made recordings of multimedia content of vital importance in different areas. In order to guarantee the integrity of these files and that they are considered valid, this Master's Degree Final Project proposes a prototype composed of a camera and a Raspberry Pi. The Raspberry Pi will process the video from the camera to live broadcast it via a web server. This video will be broadcast in an adaptive bitrate format to improve video consumer's quality of experience. Thanks to a blockchain-based system, a series of processes will analyze the generated video segments, storing summary information of its content in an immutable way. In this way, this project will try to test if registering the audiovisual content in real time inside the blockchain is feasible. The aim of this solution is to prevent the video composition from being modified by shortening the time between the broadcast and its registration as much as possible. With this method it is possible to check, in real time or later, if a video has been altered since its broadcast.

Índice

1. Introducción

Hoy en día consumimos y capturamos una gran cantidad de contenido multimedia, ya sea vídeo, audio, o ambos en un mismo archivo. La cantidad de información que se genera a diario aumenta con el paso del tiempo, dado que el avance tecnológico trae consigo la captura y almacenamiento de datos.

Esta cantidad de datos sigue un crecimiento exponencial, tal y como se recoge en diferentes predicciones y análisis. La empresa IDC (International Data Corporation) estimaba en 2018 un almacenamiento global aproximado de 33 zettabytes y predice que para 2025 será de 175 ZB[1]. Para 2025 también se estima que el 30% de los datos serán datos en tiempo real, lo que supone duplicar la cifra de 2017[1].

Cisco, en uno de sus informes anuales, estima que en 2023 las comunicaciones M2M (machine to machine) supondrán el 50% del total de los dispositivos y conexiones en la red[2]. Dentro de este colectivo se engloban diferentes dispositivos IoT (Internet of Things), entre los cuales se encuentran las cámaras de vigilancia inteligentes.

En el ámbito privado, cada vez más personas tienen acceso a dispositivos con la capacidad de capturar vídeo. A medida que el mundo de la tecnología avanza, lo hace la calidad del vídeo que pueden capturar los dispositivos, y, por tanto, la cantidad de espacio que ocupan. La mayoría del contenido que captura una persona se caracteriza por ser contenido de entretenimiento, pero también existen dispositivos que pueden capturar vídeo con finalidades completamente distintas.

Un ejemplo sería las cámaras de video a bordo del coche o *dashcam*. Estos dispositivos son capaces de grabar el trayecto de un vehículo desde diferentes puntos de vista, siendo el más común el frontal del vehículo. Estas cámaras, además de grabar vídeo como función básica, a menudo disponen de sensores que recogen información como la fecha, la velocidad del vehículo, la fuerza g y la ubicación. Los archivos que generan estas cámaras se utilizan como una forma de vigilancia inversa, de manera que en un tribunal podrían aportar pruebas contra corrupción policial o fraude de seguros. Estos dispositivos están regulados de diferentes formas en cada país. Por ejemplo, en Rusia y Alemania se aceptan pruebas en un tribunal provenientes de este tipo de cámaras, mientras que en Suiza y Austria su uso está prohibido[3].

Fig. 1: Dashcam a bordo de un coche

En España se permite su uso (sin manipulación durante la conducción) pero no queda reflejado en el Boletín Oficial del Estado de manera explícita cómo se regula. La Agencia Española de Protección de datos recoge en su informe 0456/2015[5] diferentes ocasiones en que se han tratado este tipo de datos y las limitaciones que tienen. El informe recoge un artículo del BOE donde se autoriza a Cuerpos y Fuerzas del Estado a utilizar dispositivos de captura de vídeo. Además, el informe concluye que mientras se garantice el cumplimiento del principio de proporcionalidad, el uso de estos datos es válido ante un tribunal. Este principio consiste en que las imágenes se utilicen para el propósito por el que fueron grabadas. No es válida toda la información captada de este modo: "*De conformidad con el artículo 4 de la Ley Orgánica 15/1999 de 13 de diciembre, de Protección de Datos de Carácter Personal, las imágenes sólo serán tratadas cuando sean adecuadas, y no excesivas en relación con el ámbito y las finalidades determinadas, legítimas y explícitas, que hayan justificado la instalación de las cámaras o videocámaras*" [5].

En el ámbito empresarial, se implementan sistemas de video vigilancia, los cuales, son de gran ayuda para preservar la seguridad de los establecimientos y la integridad física de los trabajadores.

En el ámbito gubernamental conocemos dispositivos de control y vigilancia como cámaras de seguridad vial y cámaras de seguridad que vigilan las calles y edificios públicos. Más recientemente en algunos vehículos de los cuerpos y fuerzas de seguridad del estado se han instalado dispositivos como cámaras de vídeo, los cuales ayudan a tener documentada una detención, persecución o delito. Además, dispositivos como drones son útiles a la hora de vigilar una zona de manera telemática, a una localización remota, sin tener que desplazarse físicamente. En el momento que escribo esto se están utilizando para controlar la pandemia SARS-CoV-2 y asegurar que la población cumple con la cuarentena. Incluso, en algunos países estos drones tienen fines sanitarios, ya que incorporan un medidor de temperatura corporal, indicando si el ciudadano al que graban presenta síntomas[4]. Resulta especialmente interesante para no exponer a agentes de la ley ante el virus.

Tanto las *dashcam*, como sistemas de video vigilancia inteligentes forman parte del colectivo de dispositivos IoT que para 2023 se estima que alcancen la cifra de 14,7 mil millones de conexiones[2]. La cual, en comparación a las 6,1 mil millones de conexiones[2] que se midieron en 2018, hace ver cómo va a aumentar tanto el número de dispositivos, como el tráfico en la red. El aumento de este tipo de tráfico contribuye a que los operadores de la red estén continuamente adaptando el ancho de banda y la velocidad a los requerimientos en constante crecimiento.

El avance tecnológico, además de traer consigo beneficios, genera vulnerabilidades de manera inherente. Por este motivo, un archivo de vídeo puede sufrir multitud de alteraciones y modificaciones, haciendo que se ponga en duda si el vídeo es realmente el original.

Existen métodos de edición de vídeo muy sofisticados, que a simple vista harían imposible diferenciar el original del alterado. La transmisión de información a través de canales inalámbricos no seguros hace que la información pueda ser interceptada, y quede expuesta, de manera que un tercero pueda realizar modificaciones.

A la hora de querer utilizar un documento audiovisual como prueba de un hecho, debemos demostrar que el vídeo no ha sido alterado ni modificado, y si es posible, asegurar que se ha grabado en un espacio temporal concreto.

A lo largo de los años se han implementado mecanismos que solventan este problema. Uno de ellos es firmar el documento de manera 'invisible'. Otros sugieren analizar la composición del vídeo para detectar distorsiones o modificaciones.

A nivel de almacenamiento, es habitual que se guarde la información de manera centralizada, por lo que está sujeta a modificaciones de manera malintencionada en un mismo nodo. A la hora de transmitir un vídeo en directo es imprescindible cerciorarse de que sigue un procedimiento que asegure su integridad desde su captura hasta su almacenamiento, pasando por la emisión y reproducción.

Blockchain se caracteriza por ser una red distribuida de usuarios que contribuyen al funcionamiento de la cadena de bloques. Esto quiere decir que, gracias a esta tecnología, cierta información del vídeo puede introducirse instantáneamente después de que el dispositivo IoT lo haya procesado. De este modo, se reduce a cero la posibilidad de que sea alterado o modificado.

1.1 Motivación

Debido al previamente nombrado aumento del uso de dispositivos inteligentes para capturar archivos de vídeo, resulta interesante aplicar tecnologías tan prometedoras como blockchain a nivel de ciberseguridad, para garantizar la integridad de un vídeo.

El mundo de las telecomunicaciones avanza a una velocidad vertiginosa, lo que requiere de continua actualización del tratamiento de los datos para mantener la seguridad.

En un momento como este, en el que tecnologías como el 5G emergen para reducir la latencia en la red, un sistema como el propuesto, tiene una alta probabilidad de funcionar en el ecosistema de las comunicaciones.

Cada vez más elementos están conectados a la red, una cámara no deja de ser otro sensor más en el océano de dispositivos IoT que se tiene previsto conectar. Por lo que la seguridad cobra un papel de gran importancia, ayudando a evitar posibles ataques.

2. Objetivos del TFM

El objetivo de este proyecto es diseñar un método que consiga grabar y emitir vídeo en directo, a la vez que preservando la integridad del vídeo lo más próximo a tiempo real que sea posible. A lo largo del mismo se evaluarán y pondrán a prueba diferentes enfoques que pretenden realizar esta tarea de la manera más eficiente posible.

Para obtener el resultado deseado es necesario investigar a fondo el funcionamiento de los sistemas de grabación y emisión de vídeo en directo existentes, así como las soluciones que garantizan la integridad de la información.

Conocer el estado del arte de ambos procedimientos ayudará a entender hasta dónde se puede llegar a desarrollar una solución.

En este proyecto se diferenciarán dos ámbitos sobre los que investigar:

El primero de ellos consiste en analizar a fondo los archivos de vídeo generados por una cámara de vídeo, así como los métodos de grabación y transmisión que usan y mecanismos de reproducción.

Por otro lado, el segundo de ellos será comprender cómo funcionan los sistemas de blockchain y cómo aplicarlos a una solución concreta. Esto implicará hacer uso de criptografía y realizar un estudio intensivo de las técnicas usadas en ciberseguridad para mantener la integridad de la información.

El objetivo final consistiría en proveer a usuarios de la capacidad emitir vídeo de manera segura, con un dispositivo tan sencillo como una Raspberry Pi y una cámara conectada. Además, no sólo en la parte de la emisión, sino también en recepción, se pretende desarrollar una manera de verificar y comprobar que la información recibida no ha sido alterada.

3. Metodología de trabajo del TFM

Para desarrollar este trabajo, se dividen las tareas a realizar en diferentes etapas.

La primera, claramente marcada por la búsqueda de información y formación académica sobre los dos ámbitos previamente nombrados (emisión de vídeo y blockchain).

Esta primera fase ha consistido en comprender la tecnología de la que se iba a hacer uso y a su vez conocer las soluciones existentes y su funcionamiento.

Una vez se ha hecho esa primera toma de contacto, el trabajo pasa a la fase de experimentación. Dado que la tecnología no es demasiado madura todavía y más si hablamos de blockchain, no existe suficiente documentación como para seguir un procedimiento claro desde el principio. Esta característica del proyecto ha tenido en consecuencia que se hayan seguido diferentes métodos de desarrollo de la solución.

Tras haber indagado y desarrollado diferentes enfoques, se procede a probarlos y evaluarlos de manera que se pueda valorar qué características tiene cada uno y qué finalidad podrían desempeñar en una solución final, aquellos que finalmente han resultado fructíferos.

4. Marco teórico

Con el objetivo de poder desarrollar una solución de la mejor manera posible, se ha contemplado adquirir información sobre las diferentes tecnologías que se van a utilizar. Por ello, resulta interesante antes del desarrollo, estudiar el funcionamiento de las herramientas involucradas.

4.1 Blockchain

¿Qué es blockchain? Desde luego, es un término que a día de hoy cualquier persona que trabaje dentro del ámbito tecnológico o más concretamente de las tecnologías de la información, ha escuchado. Lo que no está tan claro es si todos conocen qué es y como funciona.

Traducido literalmente del inglés, la palabra significa cadena de bloques. Realmente es simplemente eso, pero ¿qué son esos bloques?, ¿cómo se encadenan?, ¿por qué es necesaria para almacenar información?

Los bloques están compuestos por un hash (SHA256 p.ej.) del bloque anterior, un timestamp que indica el momento en el que se crea el bloque y datos que se pretendan almacenar en la cadena. Estos bloques están unidos entre sí gracias a la operación criptográfica del hash. De modo que todos los bloques dependen del anterior hasta llegar al primer bloque, también llamado bloque génesis. Esta característica de la tecnología hace que alterar un bloque de la cadena implique alterarlos todos. Ahí reside el potencial de esta tecnología.

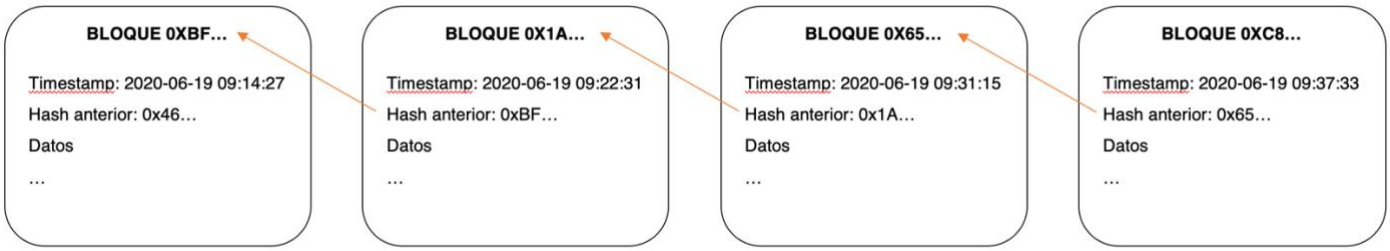


Fig. 2: Estructura de Blockchain

La primera vez que se pretendió dotar a un documento de una seguridad similar a blockchain fue en 1991[6] , cuando se presentó el proyecto "*How To Time-Stamp a Digital Document*". Este documento trata de marcar un documento con una estampa de tiempo que garantice que se creó en dicha fecha o se modificó por última vez. El propósito era evitar tratar un documento falsificado como válido. Los mismos autores publicaron más artículos en 1993 y 1997, los cuales fueron predecesores a la creación de Bitcoin.

Unos cuantos años más tarde, en 2008 se podrujo una crisis económica a nivel mundial. Una persona o grupo de personas (todavía no se conoce con certeza) publicó o publicaron un trabajo que pretendía evitar que un desastre de ese calibre volviese a ocurrir, dotando al sistema monetario de transparencia y seguridad. Se publicó bajo la autoría de Satoshi Nakamoto y en ese trabajo, a pesar de existir la tecnología, se refleja un sistema nunca visto. El proyecto se titulaba: "*Bitcoin: A Peer-to-Peer Electronic Cash System*"[7] .

Bitcoin es el proyecto más famoso basado en blockchain. Gracias a un sistema monetario virtual basado en criptografía de clave pública, dotan al sistema de la característica de no poder hacer una transacción dos veces.

La red peer-to-peer propuesta se encarga de registrar todas las transacciones basándose en *Proof-of-work*. La *Proof-of work* o prueba de trabajo es el método de asegurar que los nodos de la red no actúan de manera malintencionada. Se proponen problemas matemáticos de una dificultad proporcional a la capacidad de cómputo de la red global, de manera que se demuestre al sistema, que el nodo que está contribuyendo al minado de bloques, empeña su rendimiento en ello.

Gracias a la contribución benévola de la mayoría de los nodos, un nodo malintencionado que quisiera alterar una transacción, debería resolver una serie de operaciones matemáticas en un lapso de tiempo tan reducido, que sería imposible computacionalmente. Sólo si un nodo o conjunto de nodos consiguiesen superioridad computacional, esto sería posible. En el caso de Bitcoin existe una cantidad enorme de dispositivos que forman la red, por lo que sería algo inviable.

Los nodos hacen una especie de votación o verificación de los bloques que reciben de otros nodos. De manera que los votados como no válidos por la mayoría, se descartan y no se incluyen en la cadena de bloques, solo quedando como válidos los votados por la mayoría. Cuanta más potencia computacional se tiene, más votos se pueden enviar. Esto se conoce como algoritmo de consenso.

El coste computacional de minado de un bloque ha ido *in crescendo* desde que el Bitcoin ha ido aumentando su valor. En sus inicios, Bitcoin no tenía ningún valor, desde 2008 hasta 2010 era tan solo un proyecto al que unos pocos aficionados o *early adopters* de la tecnología se habían unido. Fue en 2010 cuando uno de los componentes de la red decidió hacer una transferencia de 10.000 BTC a cambio de una pizza[8]. A día de hoy, un BTC fluctúa entre 8 y 10 mil dólares[9].

Los “mineros” (ordenadores) que ayudan a registrar las transacciones dentro de bloques son recompensados con fracciones de la moneda (Bitcoin). Por ello se ha generado una comunidad dedicada exclusivamente al minado de criptomonedas para obtener beneficio económico a cambio.

Tras el éxito de Bitcoin, han sido muchos los proyectos que han surgido utilizando blockchain, y ha ayudado a que esta tecnología adopte diferentes formas y aporte diferentes soluciones.

4.1.1 Estructura

La clave del éxito de blockchain es sin duda su estructura, aquella que consigue dependencias de un bloque sobre todos los anteriores. El hecho de no poder alterar de manera retroactiva ningún bloque anterior, dota de inmutabilidad a los sistemas basados en blockchain.

Además, la estructura de almacenamiento es diferente a lo que se conoce tradicionalmente. Internet es una tecnología que está descentralizada, esto quiere decir que hay ciertos nodos que proveen la información y la caída de uno de ellos no supone la pérdida de dicha información. En el caso de blockchain la información está descentralizada y distribuida. Esto quiere decir que todos los nodos almacenan parte de la cadena, o en algunos casos (nodos completos), toda la cadena.

4.1.1.1 Bloques

La unidad fundamental de blockchain es el bloque. Este contiene información, en el caso de las criptomonedas, de transacciones. Los bloques incluir información de la fecha y hora exacta de creación del bloque y además debe incluir un hash que haga referencia al bloque anterior. Opcionalmente se pueden almacenar datos que se consideren importantes dentro del bloque. Se obtiene un resumen o hash a partir de estas transacciones y se almacena en un árbol de Merkle, desde donde podrá ser consultado a posteriori.

4.1.1.2 Árbol de Merkle

Un árbol de Merkle es una estructura de verificación de hashes que pretende ser más efectiva que una búsqueda sobre una base de datos al uso, para el caso de una cantidad considerable de datos. Cada hoja del árbol es un hash, siendo las hojas más alejadas de la raíz los hashes de los bloques de la blockchain. A partir de estas, se calculan hashes combinados de dos hojas anteriores, reduciendo a un logaritmo en base 2 del total de las transacciones, el número de búsquedas que hay que hacer para llegar a la raíz.

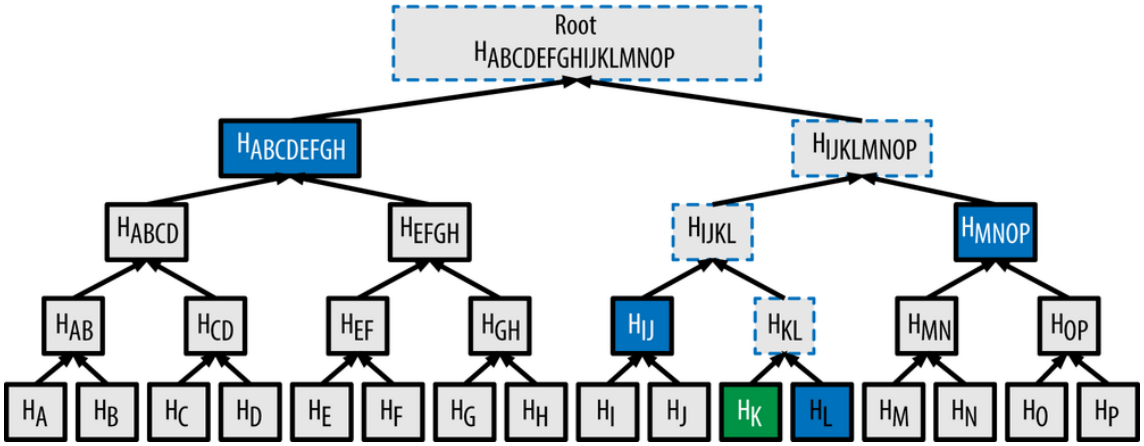


Fig. 3: Árbol de Merkle

Por ejemplo, para verificar la transacción con el hash H_K no necesitamos ir hasta H_A para comprobarlo. Simplemente conociendo H_L, H_{IJ}, H_{MNOP} y $H_{ABCDEFGH}$ podemos llegar hasta la raíz del árbol y verificar que H_K es correcto. En este ábol de 16 hojas 'bloque', necesitamos $\log_2(16) = 4$ hashes del árbol para verificar un bloque. Del mismo modo que en una blockchain, alterar un bloque provocaría un cambio en todos los bloques siguientes, en el árbol de Merkle, alterar un hash, cambiaría todas las hojas de los nodos superiores hasta la raíz[10].

4.1.1.3 Versiones

En ocasiones, la blockchain tiene diferentes estados simultáneos para los mismos bloques, ya que se minan de manera diferente en diferentes nodos. Las blockchain tienen mecanismos de voto, y el bloque minado más votado es el elegido para formar parte de la blockchain, eliminando los duplicados existentes. Esto garantiza que a los nodos siempre les va a llegar la cadena más votada y van a descartar la anterior.

Por ello nunca está claro si una versión de la blockchain es la definitiva. Estas versiones son normalmente la blockchain con un solo bloque añadido, el cual se descartaría en caso de llegar una versión con más puntuación. La naturaleza de blockchain hace que disminuya de manera exponencial la probabilidad de encontrar una versión mejor conforme se añade más bloques.

4.1.1.4 Protocolos de consenso

Una vez que se conoce la estructura de los bloques, ¿qué factores influyen a la hora de incluir un bloque como válido dentro de la blockchain?

Existen diferentes métodos que capacitan a los nodos para confirmar transacciones a través del minado. Estos métodos quieren evitar que existan nodos que participen en la red de manera malintencionada, por ello se han diseñado diferentes protocolos de consenso que determinan cómo de válido es un nodo:

4.1.1.4.1 Proof of Stake (PoS):

La "prueba de participación" consiste en demostrar cuánta cantidad de dicha criptomoneda posee el nodo. En función de la cantidad, se le dotará de mayor facilidad para minar. Este concepto se basa en el hecho de que un nodo que posee criptomonedas, no va a querer boicotear el funcionamiento de la cadena de bloques.

La ventaja que ofrece es que no hace falta tanta computación para minar como con PoW y no genera una carrera tecnológica en busca que dominar la minería.

Los inconvenientes incluyen que un minero podría estar boicoteando y a su vez ganando dinero minando bloques de manera correcta (*nothing-at-stake problem*). Además, los monederos de los mineros deberían estar conectados a la red con el riesgo de ataque que conlleva (esto ya tiene solución, llamada DPoS, o prueba de participación delegada, la cual permite distribuir desde un nodo a otros nodos, participaciones y deleguen su "poder"). Finalmente, uno de los ataques que se podrían dar bajo este protocolo de consenso es el del 51%. Un nodo que posea más de la mitad de las monedas podría mantener para siempre sus privilegios minando sus bloques, añadiéndolos a la blockchain y obteniendo la recompensa.

4.1.1.4.2 Proof of Work (PoW):

Este sistema de consenso tiene el fin de incentivar a los nodos que mayor capacidad computacional tengan. Es decir, se dificulta el proceso de minado para evitar que la capacidad de computación se destine de manera malintencionada, ya que la mayoría de nodos busca la recompensa y evitará que los intentos de alteración de la cadena se lleven a cabo. Las operaciones matemáticas propuestas son muy complicadas del lado del nodo minero, pero muy sencillas de verificar del lado del servidor.

La ventaja que ofrece es que limita los ataques de denegación de servicio o de alteración de la información.

Los inconvenientes que presenta son que cada vez es más complejo obtener beneficio del minado, creando empresas que se dedican exclusivamente a fabricar dispositivos de minado, que consumen cantidades altísimas de energía. Tal es el auge, que se han formado empresas en países como China, donde se llenan naves industriales de dispositivos de minado con el único fin de obtener criptomonedas a cambio como recompensa. Dada la fluctuación de las criptomonedas, en épocas de valores mínimos, estos mineros han preferido no minar debido a que lo que obtenían de recompensa era menor a los gastos generados pos la planta de minado.

Este método es el más ampliamente utilizado. Se usa en Bitcoin, Ethereum y otras monedas.

4.1.1.4.3 Proof of Authority (PoA):

El mecanismo de "prueba de autoridad" pretende denominar a ciertos nodos como "validadores" de bloques. Esto quiere decir, que de manera arbitraria se escoge a los nodos "de fiar". Esto implica que los nodos designados como validadores se mantengan lo menos comprometidos como sea posible. Se incentiva a los nodos a seguir siendo validadores y no perder los privilegios. El sistema debe poder eliminar a los nodos que no mantengan una buena reputación.

Este método elige de manera aleatoria a los validadores y no permite verificar más de un bloque seguido al mismo nodo.

La ventaja es que usa poca energía a la hora de minar bloques, por lo que es más escalabre y sostenible que el resto de algoritmos de consenso.

La desventaja que presenta es la descentralización, lo cual no caracteriza al concepto de blockchain. Además, la vulnerabilidad de los nodos puede ser crítica a la hora de mantener el correcto funcionamiento de la cadena. Este método, a menudo se lleva a cabo en blockchains privadas y no tanto en públicas.

Existen otros métodos como *Proof of Burn* (PoB) o *Proof of Importance* (PoI), menos utilizadas y con similitudes a las anteriormente descritas.

4.1.2 Tipos

Bajo el concepto blockchain se encuentran diferentes variantes como pueden ser las Blockchain públicas o privadas, es decir, las que requieren de permiso para participar y las que prescinden de él, además de soluciones híbridas.

4.1.2.1 Públicas

Las blockchain públicas, sin permiso o *permisionless* son, desde el punto de vista de Satoshi Nakamoto, una red sin autoridad central que restrinja el acceso al sistema. Existe libertad completa para formar parte de la red. Las condiciones para minar están escritas en el código del software y no se pueden alterar. La manera de controlar las acciones malintencionadas es utilizando el previamente explicado mecanismo de *Proof of Work*.

A su vez, una red pública puede ser abierta (Bitcoin, Ethereum, casas de apuestas, videojuegos…) o cerrada (votación por parte de una cantidad de personas limitada, es decir, ciudadanos de dicha ciudad, comunidad o país).

4.1.2.2 Privadas

Por otro lado, las blockchain privadas, que requieren de permiso o *permissioned* son aquellas que añaden una capa de control de acceso a la red. A diferencia de las públicas, en las privadas es el propio administrador de la red el que se encarga de evaluar a los nodos, y no el consenso general.

Existen críticas a este tipo de blockchain, por parecerse más a una base de datos compleja, que a lo que el propio concepto hace referencia. La desventaja principal es que existe un nodo que puede ser comprometido y controlar el 100% de la red. Cosa que en una red pública no puede ocurrir, ya que, comprometiendo un nodo, obtienes una fracción infinitesimal de computación.

Las redes privadas pueden dividirse como las públicas en cerradas (asuntos militares, poder legislativo, agencia tributaria, ministerio de defensa…) que requieren de máxima restricción, o en abiertas (cadena de suministro, registros de gastos a nivel gubernamental, ganancias o transacciones corporativas) las cuales estarían abiertas a los nodos de una empresa u organismo, pero de manera controlada.

4.1.3 Transacciones

Dado que el mayor enfoque de la tecnología blockchain son las criptomonedas, cabe destacar el funcionamiento de las transacciones. Para comprender como funcionan, debemos saber que existen herramientas llamadas carteras, donde es posible almacenar las direcciones y claves de manera segura. Existen carteras que generan claves de manera aleatoria llamadas JBOK (Just a Bunch of Keys) y otras que las generan a partir de una semilla, llamadas deterministas. [10]

La manera más común es en forma de software. Este método permite tener más de una cartera para poder dividir la totalidad de las monedas que se poseen y para poder hacer transacciones de manera anónima.

Dado que se puede llegar a almacenar una gran cantidad de dinero dentro de una cartera de software, que a su vez estaría dentro de, por ejemplo, una aplicación móvil, resulta interesante no almacenar el total de las monedas bajo el mismo software, ya que, si llegase a estar comprometido dicho software, lo estaría también la totalidad de las criptomonedas almacenadas.

Para evitar correr el riesgo, existen carteras que se mantienen offline, con el fin de almacenar la mayor parte de las monedas y dejar en forma de software sólo las que vayan a utilizarse a menudo. Esta cartera, debería conectarse a la red para transferir fondos a la que se utilice de manera más frecuente, por ejemplo creando varias carteras con fondos, asegurando, como se comentaba anteriormente, la privacidad.

En el caso de Bitcoin, una transferencia se realiza habitualmente desde una cartera donde se sitúan los fondos y existen dos destinatarios. Uno de ellos sería la cartera a la que va destinado el pago y la otra sería la misma que envía el dinero, para recibir el "cambio".

Sería como pagar con un billete de diez euros en un establecimiento, comprar un producto que vale dos euros y recibir los ocho del cambio.

Es decir, en Bitcoin, se transfiere la totalidad de dinero que existe en la tarjeta para recuperar el "cambio" si lo hubiese. Es algo habitual, ya que solemos tener más dinero del que vamos a pagar por un producto. No es habitual que sea la misma cantidad.

Las carteras de Bitcoin son capaces de reunir diferentes fuentes de monedas, llamadas *UTXO* (*unspent transaction outputs*), las cuales son remanentes de transferencias anteriores, para completar un pago de mayor cantidad e ir eliminando *UTXO*s igual que una persona intenta deshacerse de las monedas de menor valor que lleva en el bolsillo.

Además es posible añadir más destinatarios en una misma transacción.

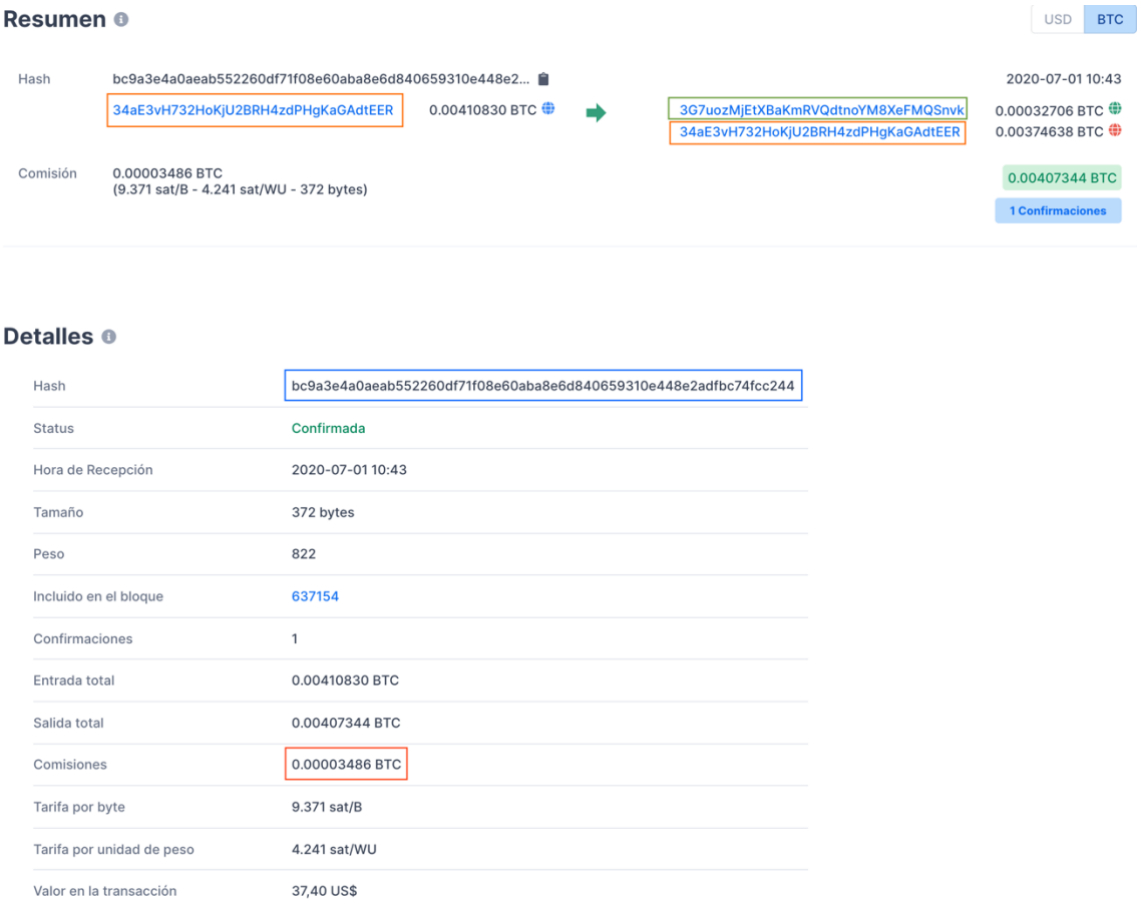


Fig. 4: Transacción Bitcoin

Para clarificar la explicación, se ha tomado una captura de una operación aleatoria de Bitcoin, real, desde la web Blockchain.com (que, aunque tenga el mismo nombre de la tecnología, es una empresa privada dedicada al trading de criptomonedas).

Se puede apreciar en la Figura 4 que la dirección rodeada por un cuadrado naranja aparece dos veces, tanto en la parte del emisor como del receptor. La verde sería la que recibe la transacción. Además, existe un hash único, rodeado de azul, para esa transferencia de monedas, irrepetible e inmutable. Como añadido, vemos la comisión aplicada a esa transferencia rodeada de rojo. Esta es la cantidad de la transferencia que se destina a los mineros como recompensa, por conseguir minar los bloques.

Ethereum funciona más parecido a lo que ya conocemos, es decir, la transferencia se hace de A a B sin tener que transferir la totalidad que hay en A.

4.1.4 Smart Contracts

Además de transacciones entre direcciones, existen métodos de automatizar pagos y registrarlo en la blockchain. Los Smart Contracts son una serie de instrucciones programadas dentro de la blockchain que, siguiendo una serie de condiciones, ejecutan transferencias de fondos. Este tipo de contratos resulta de interés dado que el contrato es inmutable y, por lo tanto, las partes interesadas en el contrato, confían en su correcto funcionamiento.

Este tipo de contratos pueden visualizarse, por lo que ofrecen transparencia al usuario. De alguna manera, toda la burocracia existente en las empresas de hoy en día referente a los contratos, desaparecería.

Es un intermediario entre dos entidades. Si Alice quiere comprarle un kilo de manzanas a Bob, sencillamente envía una transferencia de dinero al Smart Contract, el cual, liberará la transacción hacia Bob, una vez Alice confirme la llegada de las manzanas.

Desde las webs de seguimiento de transferencias de criptomonedas, podemos distinguir entre transferencias entre dos direcciones, como en la figura 4, y las transferencias producidas por un Smart Contract:

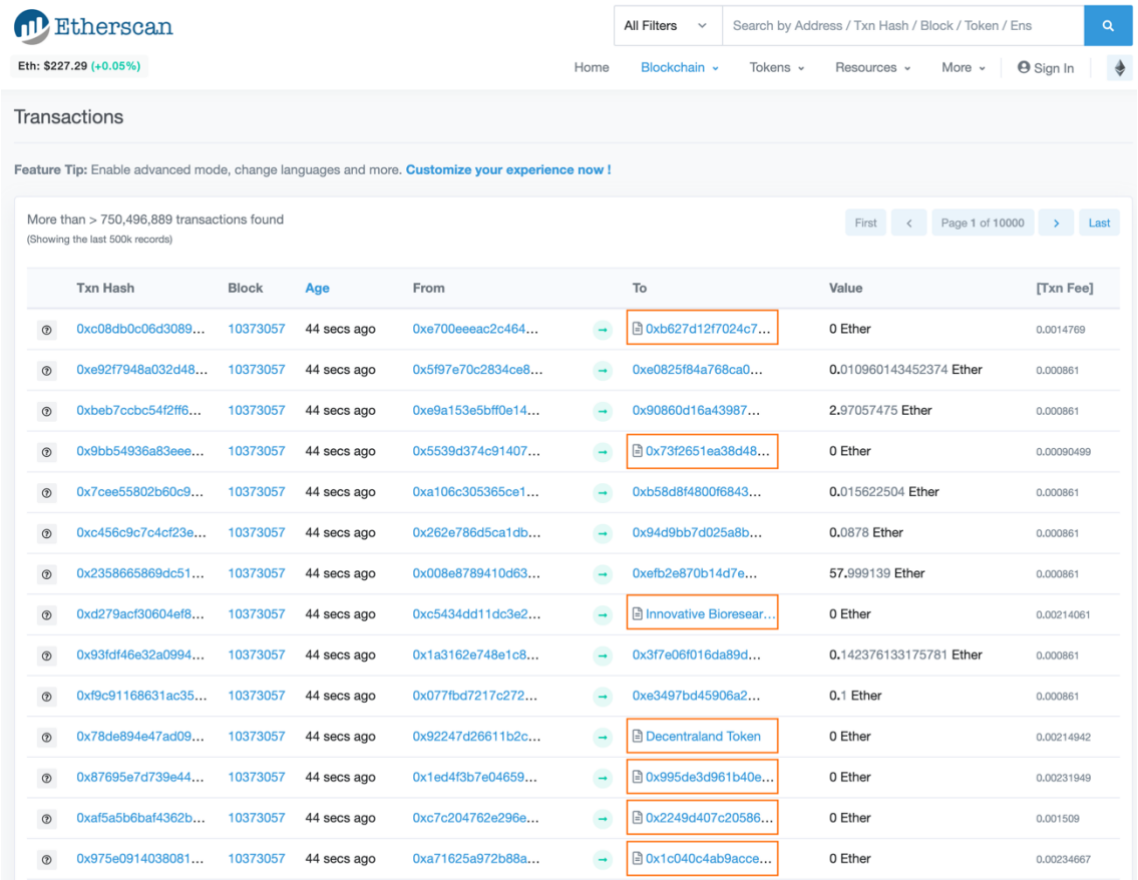


Fig. 5: Lista de transacciones de la red Ethereum en un momento concreto [11]

En la Figura 5 podemos observar una lista de transacciones donde se aprecia cuales van asociadas a un Smart Contract por el símbolo que aparece antes de la dirección del destinatario. Si hacemos *click* en una de ellas, nos muestra más información:

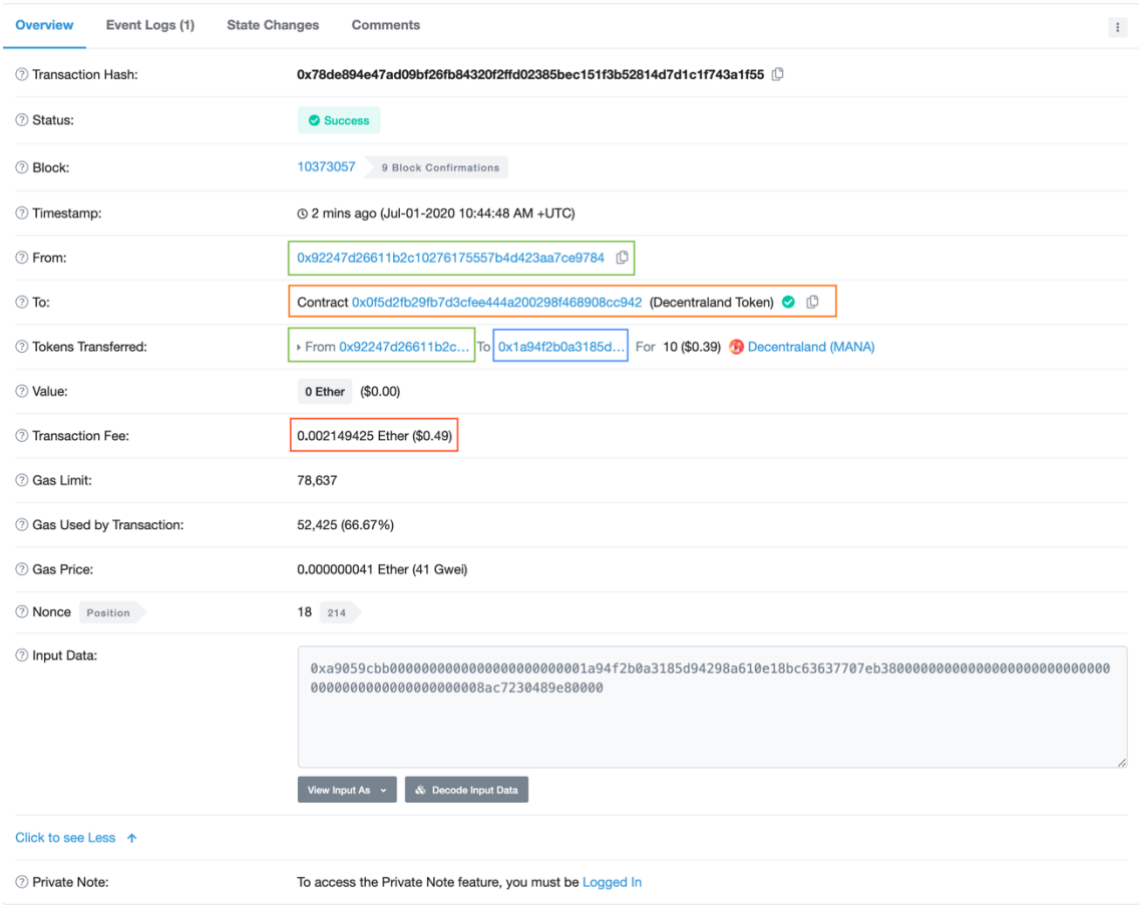


Fig. 6: Transferencia asociada a un Smart Contract

Como se puede apreciar en la Figura 6, existe un emisor (encuadrado en verde), que a través de un contrato (encuadrado en naranja), envía una transacción hasta un destinatario (encuadrado en azul) pagando una comisión (encuadrado en rojo).

Cabe destacar que los contratos en Ethereum siguen un estándar llamado *token ERC-20*. Es interesante desarrollar un mismo tipo de contrato para que exista compatibilidad entre ellos, ya que a priori, simplemente es un programa que se escribe en un lenguaje y se ejecuta en la plataforma blockchain.

El lenguaje más ampliamente utilizado para programar Smart Contracts es Solidity. Es un lenguaje de programación orientado a objetos, de alto nivel. Es el utilizado en las redes de Ethereum, diseñado para funcionar en EVMs (*Ethereum Virtual Machines*). Tiene similitudes con C++, Python y JavaScript.

4.2 Emisión de vídeo

Dado que la finalidad y objetivo de este trabajo es asegurar la integridad de archivos de vídeo, cabe hacer un estudio intensivo de la composición interna de los archivos de vídeo, así como métodos de grabación y reproducción en tiempo real. Este estudio servirá para, en el momento de desarrollar la herramienta, poder elegir la alternativa adecuada.

4.2.1 Métodos y técnicas de emisión

Grabar vídeo y emitirlo en directo requiere de herramientas diferentes a la grabación de vídeo habitual. El hecho de ser en directo implica que el vídeo no se envíe al completo, ya que sino, el receptor del vídeo tendría que esperar un tiempo igual a: el tiempo de duración del vídeo más el tiempo de transmisión a través de la red. Por supuesto esto es impracticable si el propósito es consumir el contenido multimedia lo más cercano al tiempo de emisión.

Es por esto por lo que, existen métodos y técnicas de grabación que consiguen fragmentar el vídeo en el momento de la grabación para poder reproducir poco a poco el vídeo en el destino desde el inicio de la grabación. De este modo se consigue, a cambio de un pequeño retraso, emitir lo grabado instantes después en remoto.

Hoy en día, plataformas de *streaming* de contenido audiovisual como pueden ser series, películas o simplemente vídeos realizados por cualquier persona, implementan un sistema de codificación a diferentes calidades. Este sistema permite que, en función del ancho de banda del usuario receptor del vídeo, se descargue el vídeo a mayor o menor calidad.

Conforme aumenta el auge de las emisiones en directo, se comienza a implementar también este tipo de técnicas en el ámbito del *live streaming*. Por ello, en este proyecto puede aportar un valor añadido a la hora de mejorar la QoE (*Quality of Experience*) del usuario. Es decir, implementando este sistema de emisión de vídeo, el usuario podrá experimentar ligeros cambios en la calidad del vídeo, pero no experimentará paradas en la reproducción. Un estudio que analizó 400.000 visualizaciones en YouTube concluyó que el ser humano prefiere ver un pequeño fragmento de vídeo a menor calidad que esperar a que cargue el de alta calidad (*rebuffering*). [12]

Definir la longitud de los segmentos a emitir no es tarea fácil ni trivial, sino que depende del entorno en que se trabaje.

Los fragmentos cortos resultan útiles para conseguir una más rápida adaptación del *bitrate* y se disminuye el riesgo de que el vídeo se pare, ya que ocupan poco espacio y tardan poco en transmitirse.

Por otro lado, los fragmentos largos son sinónimo de mejor eficiencia de codificación (menos repetición de cabeceras al contener más información en un mismo archivo), lo cual, en función de la(s) calidad(es) que se quiera(n) grabar y el espacio o capacidad de almacenamiento que se disponga, puede interesar.

Una prueba [13] dedicada a analizar diferentes longitudes de fragmentos de vídeo en formato adaptativo, concluye que la reducción de tamaño de los fragmentos supone una pérdida aproximada de 1,5 dB de PSNR (*Peak Signal-to-Noise Ratio*) comparando fragmentos de 1s y de 15s. O lo que es lo mismo, se reduce el margen entre la señal y el ruido de la información, causando lo que se comentaba anteriormente, pérdida de eficiencia de codificación.

Además, esa prueba analiza en dos entornos de conexión diferente (HTTP 1.0: no persistente y HTTP 1.1: persistente) la emisión de fragmentos de 1, 2, 4, 6, 10 y 15 segundos y mide que cantidad de información efectiva se transmite por segundo. De este modo obtiene una gráfica que muestra cómo se comporta la longitud respecto a la eficiencia de transmisión.

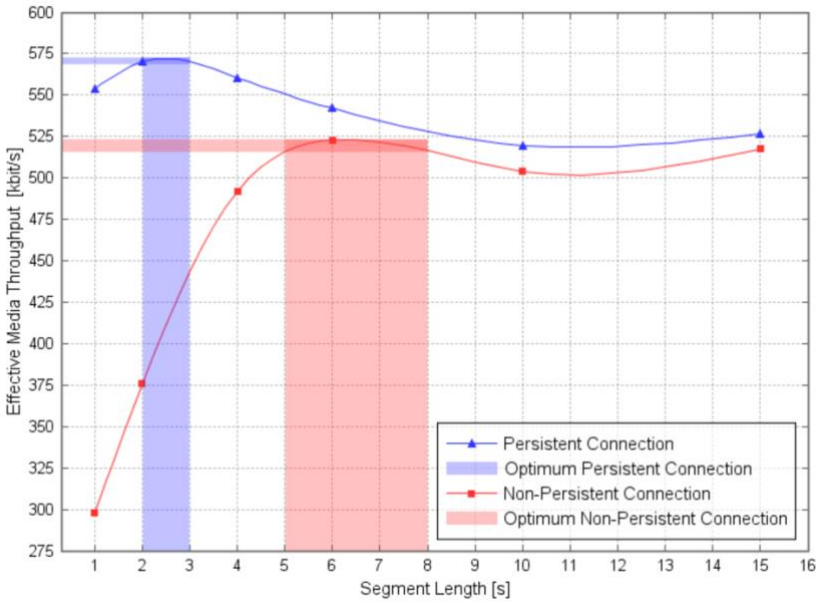


Fig. 7: Eficiencia de transmisión de video respecto a la longitud de fragmentos y tipo de conexión HTTP

El análisis concluye que el tamaño idóneo para utilizar en este tipo de transmisiones debe rondar entre 2 y 4 segundos, sin ser demasiado corto como para sobrecargar la red a peticiones y generar demasiadas cabeceras y sin ser demasiado largo como para tener que esperar demasiado tiempo para la descarga o cambio de calidad.

Una vez conocemos qué procedimiento seguir para grabar vídeo preparado para ser emitido, debemos conocer qué formatos existen y se aplican en la actualidad a proyectos reales.

4.2.1.1 MPEG-DASH

MPEG-DASH es una técnica de emisión de vídeo de *bitrate* o tasa de bits adaptativo desarrollada por MPEG (*Moving Picture Experts Group*) diseñada para funcionar sobre servidores HTTP. De hecho, DASH viene del acrónimo de *Dynamic Adaptive Streaming over HTTP*, que significa Emisión Adaptativa Dinámica sobre HTTP.

Es la primera solución para emitir sobre HTTP que se reconoce como estándar internacional.

Existe una organización llamada DASH-IF (*Dash Industry Forum*) compuesta por empresas como Microsoft, Netflix, Google, Samsung y Adobe entre otras, que se dedica a dar directrices de cómo implementar esta técnica para los diferentes casos de uso.

Su manera de funcionar es generando un segmento inicial, segmentos de vídeo con contenido multimedia dentro y un archivo llamado Media Presentation Description, encargado de reproducir el contenido a partir de los segmentos.

Un reproductor DASH simplemente necesita conocer la ubicación del archivo MPD para poder reproducir los segmentos, ya que en la composición del MPD está la estructura de grabación y las guías para la reproducción.

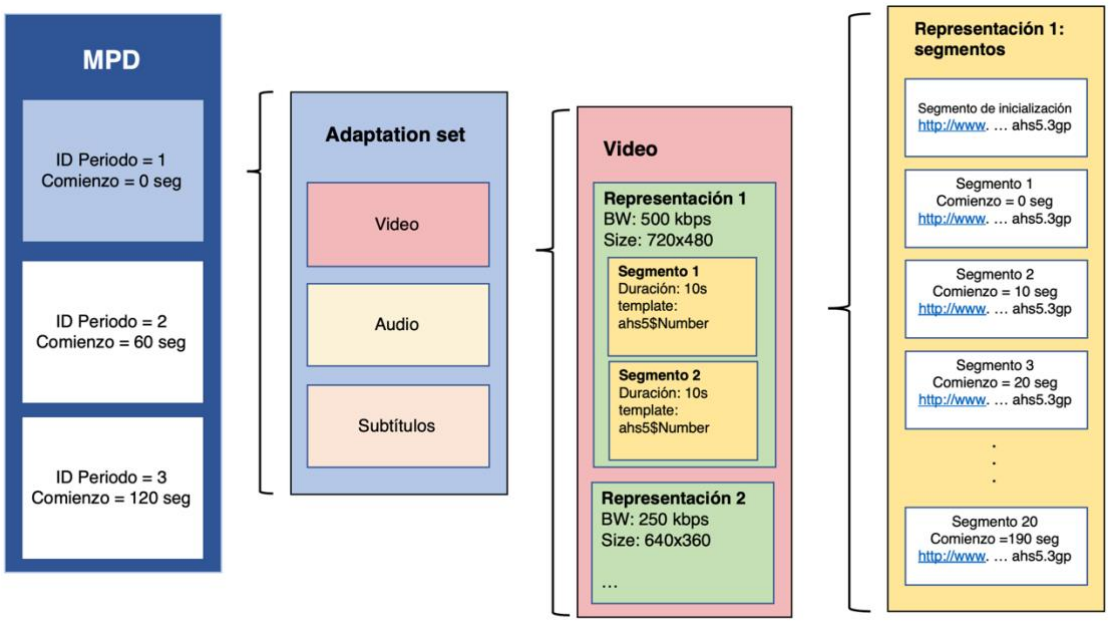


Fig. 8: Estructura del archivo *Media Presentation Description*

En la Figura 8 se muestra un esquema de cómo funciona la estructura interna del archivo *MPD*. En primer lugar, se subdivide en periodos de tiempo. A su vez, estos intervalos de tiempo contienen diferentes calidades de vídeo en su interior. Cada calidad, a su vez, contiene una representación del vídeo (una representación contiene, y ese segmento de vídeo contiene información de tiempo de reproducción en su estructura interna. Es una especie de estructura en árbol que optimiza la búsqueda de información a la hora de reproducir el vídeo.

El formato de los segmentos es .mp4, como no podía ser de otro modo al tratarse de una solución desarrollada por MPEG.

4.2.1.2 HLS

HLS o *HTTP Live Streaming* es un protocolo de comunicaciones desarrollado por Apple con el propósito de conseguir reproducir vídeo en directo de manera adaptativa sobre HTTP.

Es similar a DASH, ya que ambos utilizan fragmentos de vídeo para la reproducción en directo. Además, HLS se basa en HTTP como DASH. De este modo, a diferencia de otros protocolos como RTP, el tráfico se trata como una petición HTTP normal y no como transferencia de archivos en sí.

El protocolo HLS está programado para funcionar en entornos de Apple como puede ser Safari o cualquier aplicación de Apple OS e iOS.

La estructura de los archivos es similar a la del *Media Presentation Description* de DASH. Existen diferentes calidades y para cada calidad varios segmentos que permiten la adaptación entre calidades. En HLS existe un archivo "Máster" análogo al *MPD* de DASH y archivos de tipo *playlist* o lista de reproducción, que contienen los segmentos de vídeo a reproducir en diferentes calidades.

4.2.1.3 Generación de segmentos: FFmpeg

Conociendo las alternativas que existen para emitir vídeo en directo de manera adaptativa, debemos encontrar una herramienta que sea compatible con ambos formatos para poder comparar su funcionamiento. FFmpeg es una herramienta de grabación, codificación y decodificación de vídeo que puede utilizarse para grabar vídeo y emitirlo en directo.

Es compatible con MPEG-DASH y con HLS, por lo que es perfecta para el proyecto que se va a desarrollar.

A pesar de estar desarrollado en GNU/Linux, compatible con la mayoría de sistemas operativos.

4.2.2 Reproducción

La reproducción de contenido multimedia en formato adaptativo es capaz de decidir qué fragmentos descargar, o no, en función del estado de la red a cada momento. Dinámicamente, el reproductor toma la decisión de qué descargar a continuación.

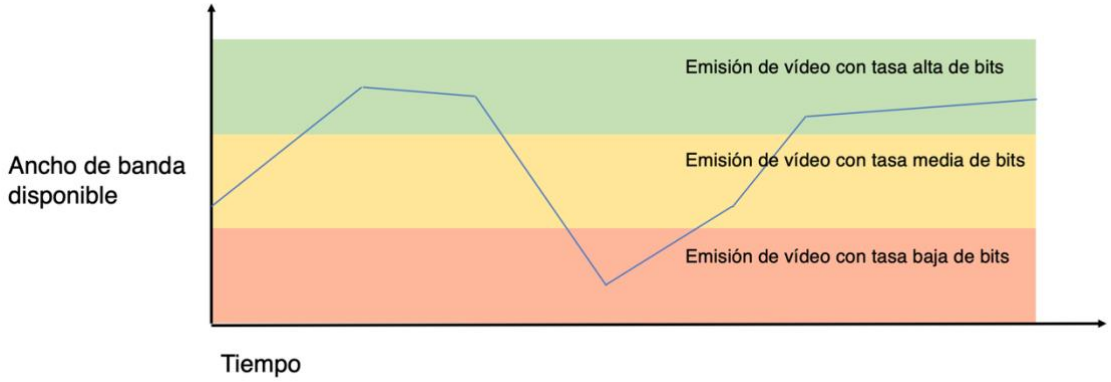


Fig. 9: Representación del funcionamiento de la emisión de vídeo adaptativa

En la Figura 9 se aprecia que, al variar el ancho de banda disponible en la parte del cliente que visualiza el vídeo, se descargan fragmentos de vídeo de mayor o menor calidad, en función de la capacidad de la red.

Tras haber estudiado los diferentes formatos y técnicas existentes de emisión y las diferentes soluciones de grabación y obtención de segmentos, es momento de analizar los reproductores que existen y ofrecen compatibilidad con los vídeos fragmentados de calidad adaptativa.

Basándonos en la popularidad de las soluciones existentes, debemos conocer principalmente dps de ellas:

4.2.2.1 Shaka Player

Shaka Player es un reproductor diseñado para reproducir contenido adaptativo, de código abierto y basado en JavaScript. Está compuesto por una serie de librerías y está preparado para ejecutarse sobre cualquier buscador. No requiere de *plugins* específicos, ya que utiliza mecanismos estándar de comunicación web como *MediaSource Extensions* y *Encrypted Media Extensions*.

Dentro de sus librerías existe una gran cantidad de métodos que diversifican el uso que se le puede dar a Shaka Player.

Shaka es un reproductor respaldado por Google, con una comunidad que da soporte en su repositorio de GitHub. Que esté continuamente actualizándose lo posiciona como uno de los referentes en cuanto a reproductores de formato adaptativo.

Es compatible con DASH y HLS, soportando vídeo en directo, vídeo bajo demanda (VOD), vídeo encriptado y diferentes estructuras internas de DASH y HLS.

4.2.2.2 Video.js

Video.js es un reproductor web de código abierto pensado para funcionar en HTML5. Además, soporta videos Flash, de YouTube y Vimeo (a través de *plugins*). Soporta la reproducción multiplataforma desde escritorio y dispositivos móviles. La propia herramienta dice estar presente en más de 400.000 webs.

Es capaz de procesar los formatos DASH y HLS, bajo diferentes estructuras.

Video.js también dispone de una comunidad de usuarios que se encarga de resolver problemas y añadir nuevas funcionalidades a la herramienta.

5. Estado del arte

Asegurar la integridad de la información forma parte de las tareas relacionadas con la ciberseguridad. Existen diferentes métodos de criptografía que ayudan a conseguir la inmutabilidad de los datos.

5.1 Emisión de vídeo

A la hora de tratar archivos de tipo multimedia como pueden ser vídeos, existen áreas de investigación dedicadas a ello. La disciplina más avanzada en este ámbito se conoce como investigación forense de información multimedia.

El análisis forense de vídeos es hoy en día una pieza clave a la hora de investigar actividades como fraude, ciberataques o distribución de contenido ilegal.

Como se comenta en el apartado de la introducción, cada vez generamos más información, y esa información tiende a ser de tipo multimedia. Esto afecta indirectamente a que, la mayoría de las investigaciones sobre delitos, impliquen el tratamiento de pruebas de tipo digital. A raíz de esto, se genera la necesidad de poder analizar las pruebas de la manera más profunda posible.

Para solucionar el problema que plantea desconocer la veracidad de una prueba, existen campos de investigación específicos[14]: identificación de la fuente de la información, identificación del entorno, identificación del tipo de contenido, recuperar información modificada, reconocer alteraciones en el contenido o identificación de fragmentos son algunos de ellos.

Para el presente proyecto resulta interesante conocer qué trabajo se realiza a la hora de detectar cambios en un archivo.

Se puede clasificar la alteración de contenido en campos diferentes:

- Falsificación por copia: conocida como clonado por haber duplicado secciones de una imagen. En una ocasión de modificó una imagen de un misil iraní, donde hicieron desaparecer los misiles para hacer creer que no se lanzaron. La falsificación se detectó utilizando diferentes técnicas como transformadas de Fourier-Mellin (FMT). [15] [16]
- Detección de retocado del contenido: para hacer que la imagen o vídeo sean más atractivos.
- Borrado parcial de determinados objetos.
- Combinación de información de diferentes imágenes.
- Manipulación de la luminancia, color o contraste.
- Manipulación de la geometría de los objetos.

En función de la profundidad con la que se trata a la información existen diferentes niveles:

- Bajo nivel: examinar píxeles y coeficientes DCT (*Discrete Cosine Transform*). Algunos métodos de falsificación de contenido multimedia alteran píxeles adyacentes a la zona modificada. Este método está relacionado con el campo de investigación de identificación de la fuente de información, ya que toman como referencia un identificador común, el SPN (*Sensor Pattern Noise*), el cual viene predefinido de fábrica por el proceso de producción del sensor.

- Medio nivel: inspeccionar el archivo para detectar bordes demasiado pronunciados, líneas demasiado rectas o zonas emborronadas de forma artificial. Además, la dirección de la luz y las sombras ayudan en esta investigación.

- Alto nivel: detección más simple, sencillamente basándose en los rostros que aparecen, las formas, y a partir de previo entrenamiento a base de datos, detectar si la combinación de los elementos es factible.

5.2 OriginStamp

OriginStamp es una empresa que ofrece el servicio que poner una estampa de tiempo a tus archivos. De modo que, mediante una transacción en Bitcoin, almacena un resumen o hash del archivo de manera inmutable.

Permiten almacenar hasta cinco estampas de tiempo por mes de manera gratuita. Además, ofrece planes d pago que van desde 10 hasta 250 dólares mensuales para poder registrar hasta 100.000 archivos al mes.

Cada registro tarda en almacenarse en la blockchain de Bitcoin diez minutos aproximadamente, dado que es el tiempo medio de minado que está programado en la red.

Para este caso de uso, diez minutos son demasiados al ser un vídeo que tratamos en tiempo real, es por ello que esta solución no se adapta a las necesidades del proyecto.

5.3 Archangel

Archangel se encarga de recolectar información y firmas digitales y almacenarlas de manera segura. Utiliza un sistema descentralizado de bases de datos (DLT) que se basa en *Proof-of-Work*. Es una red blockchain que requiere de permiso para acceder.

El enfoque principal de Archangel es mantener un registro de información gubernamental o corporativa para evitar la alteración de la información.

Esta solución no concuerda con el significado real de blockchain, dado que se tratan los datos de manera privada y el hecho de que el nodo que controla la blockchain se vea comprometido podría desencadenar en un fallo de seguridad.

5.4 Proof of Existence

Proof of Existence es un servicio que se encarga de almacenar los hashes de los archivos que quieras registrar en la blockchain de Bitcoin. Es similar a OriginStamp, pero de manera “gratuita” ya que no pagas por el servicio, pero sí por las transacciones con una dirección que tenga fondos de BTC.

Tiene el mismo inconveniente, el tiempo de minado no permite que la solución pueda considerarse de tiempo real. Por otro lado, es una solución que no lo pretende, simplemente ofrece el servicio de registrar un hash en la blockchain.

5.5 IPFS

IPFS o Sistema de Archivos Interplanetario, es un sistema de almacenamiento de datos distribuido y descentralizado. Basado en tecnologías como BitTorrent, se compone de una red *peer-to-peer* capaz de compartir y registrar información de la red.

IPFS pretende poder eliminar duplicados en su red y almacenar resúmenes únicos de archivos. Resulta interesante a la hora de elegir una base de datos, ya que ofrece la posibilidad de acceder a los datos a través del hash del archivo.

Es un proyecto que todavía se encuentra en una fase de prueba. Podría utilizarse de manera complementaria en este proyecto, ya que lo que se busca no es almacenar el vídeo completo, sino solo su hash. Con ese sistema se podrían almacenar y descargar los archivos a posteriori para poder consumir el contenido multimedia.

No sería apto para el presente proyecto dado que el prototipo ideado pretende consumir poco ancho de banda y ser lo más eficiente posible. El hecho de formar parte de esta red P2P implica estar compartiendo datos a tasas muy altas de transferencia. Además, la comunidad que lo forma es todavía muy pequeña y no cumpliría los requisitos de una blockchain.

5.6 Naivechain

Naivechain es una implementación de una solución blockchain que pretende simplificar el funcionamiento de las blockchain "en producción". De hecho, a sí mismo se define como una blockchain en 200 líneas de código.

Esta blockchain propone almacenar en un bloque: el índice o identificador de número de bloque, el hash anterior, un timestamp, información que se considere necesaria y el hash de todo lo anterior, es decir, un SHA 256 del contenido del bloque.

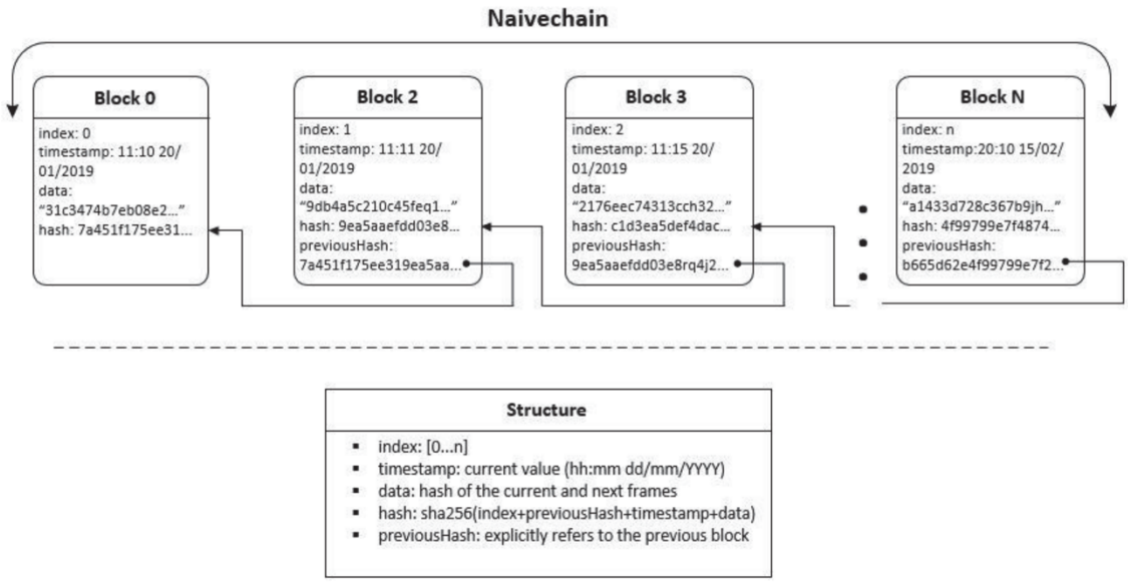


Fig. 10: Esquema Naivechain [20]

La solución Naivechain, al comienzo del desarrollo del proyecto y durante la fase de investigación, se consideró como una opción válida e iba a formar parte del desarrollo final. Más adelante, debido al desarrollo de un enfoque más concreto y específico se descartó el uso de la herramienta como tal. Aún así, se tomaron referencias del sistema para implementar la solución final.

6. Desarrollo y resultados del trabajo

Tradicionalmente se han almacenado datos de manera centralizada, más tarde las empresas de almacenamiento en la nube decidieron descentralizar sus sedes y actualmente vivimos en la época de distribuir las redes de almacenamiento de datos.

A raíz de investigar la tecnología existente y las soluciones actuales, el presente proyecto pretende poner a prueba un prototipo de grabación y almacenamiento de la información de manera inmutable, además de proveer de un sistema de reproducción y consumo del contenido multimedia con su respectiva verificación.

Todas las soluciones estudiadas y el marco teórico que las rodea han ayudado a este proyecto a definir una solución diferente para un caso de uso específico, que es el entorno de las emisiones de vídeo en directo.

6.1 Hardware

Este proyecto va a estar desarrollado en una Raspberry Pi 3B+. Se ha elegido este dispositivo por su versatilidad y compatibilidad. Además, este proyecto también tiene como objetivo analizar el rendimiento de un dispositivo tan simple, sencillo y asequible para realizar el proceso previamente explicado.

Algunas características que tiene el dispositivo son: Wifi doble banda (2,4 GHz y 5 GHz), Bluetooth 4.2, procesador quad core a 1,4 GHz, 1 GB de memoria RAM, cuatro puertos USB 2.0, puerto HDMI, ranura para tarjetas microSD, puerto minijack y puerto Ethernet RJ45. [17]

Es un dispositivo compacto por sus dimensiones: 85 x16 x 19,5 mm y 50 gramos de peso. Esto lo convierte en un dispositivo extremadamente portable.

El dispositivo de grabación será una cámara del mismo fabricante que va conectada por un cable directamente a la placa base del miniordenador. La Raspberry Pi Camera Module V2 es una cámara fabricada con un sensor Sony IMX219 de 8 megapíxeles capaz de grabar video de resolución 1080p a 30 fps o 720p a 60 fps. [18]

Fig. 11: Hardware del prototipo

6.2 Software

El software instalado en el dispositivo es la distribución de GNU/Linux pensada para funcionar en este dispositivo. Está basado en Debian, lo que garantiza compatibilidad con una enorme cantidad de software.

6.3 Grabación: FFmpeg

Para realizar la tarea de grabación, se ha escogido la herramienta FFmpeg, previamente explicada, por la versatilidad que ofrece. Esta herramienta permite modificar una generosa cantidad de parámetros a la hora de capturar vídeo, lo que hace que podamos realizar diferentes pruebas a la hora de desarrollar una solución.

Dado que es una herramienta pensada para ser ejecutada desde la línea de comandos, en este proyecto se ha elegido el módulo '*fluent-ffmpeg*' de Node.js como entorno para trabajar de manera más cómoda.

En el Anexo se detalla la información relacionada con la instalación de estas herramientas.

Tras disponer de todas las herramientas instaladas, se configura el funcionamiento de la cámara de vídeo, que, por defecto, no está correctamente configurada.

Para ello podemos introducir en la línea de comandos:

```
sudo modprobe bcm2835-v4l2
```

Para no introducir este comando cada vez que se necesite utilizar la cámara, es posible modificar en el kernel la serie de módulos que se quieren ejecutar al iniciar el sistema:

```
sudo nano /etc/modules
bcm2835-v4l
```

Para iniciar la grabación debemos consultar la documentación de FFmpeg[19] y conocer qué parámetros necesitamos en la emisión. Para este prototipo se ha elegido el siguiente conjunto de parámetros:

```
var ffmpeg = require('/usr/local/lib/node_modules/fluent-ffmpeg');

var grabacion = new ffmpeg();


grabacion.addInput('/dev/video0')

.inputOptions(['-y', '-nostdin', '-f v4l2', '-video_size 1280x720', '-framerate
30'])

.outputOptions(['-vcodec h264_omx', '-keyint_min 0', '-g 100', '-map 0:v', '-b:v
1000k', '-f dash', '-seg_duration 4',

'-use_template 1', '-use_timeline 0', '-init_seg_name init-video0-
$RepresentationID$.mp4',

 '-media_seg_name video0-$RepresentationID$-$Number$.mp4', '-utc_timing_url
https://time.akamai.com/?iso', '-remove_at_exit 0'])

.output('/var/www/html/segmentos/video.mpd')

.run();
```

En primer lugar, se ejecuta la librería *fluent-ffmpeg* y se genera un objeto llamado grabación que hace referencia a dicha librería. Lo siguiente es el comando de FFmpeg que va a ejecutarse.

Se especifica que el dispositivo de entrada es *video0*, que es como reconoce la Raspberry Pi a la cámara.

Las opciones de entrada son:

- **`-y`** : sobrescribe segmentos que existan con el mismo nombre
- **`-nostdin`** : con este comando se evita que FFmpeg trate de interactuar por consola
- **`-f v4l2`** : se especifica que el formato de entrada es Video4Linux versión 2
- **`-video_size 1820x720`** : se especifica la resolución del vídeo
- **`-framerate 30`** : se especifica la tasa de frames a grabar

Y las opciones de salida:

- **`-vcodec h264_omx`** : se especifica la codificación hardware
- **`-keyint_min 0`** : se especifica que cualquier frame puede ser tratado como key frame
- **`-g 100`** : al menos cada cien frames, uno debe ser key frame
- **`-map 0:v`** : se mapea el flujo de entrada 0 (en este caso el único) al vídeo
- **`-b:v 1000k`** : se configura el bitrate del vídeo, en este caso 1000 kbps
- **`-f dash`** : se especifica que el formato de salida va a ser DASH
- **`-seg_duration 4`** : se especifica que la longitud del segmento sea 4 segundos
- **`-use_template 1`** : se especifica que al generar los segmentos, utilice la plantilla de nombres

`-use_timeline 0` : se especifica que no incluya información de tiempo en la plantilla

`-init_seg_name init-video0-$RepresentationID$.mp4` : se especifica el nombre del segmento inicial

`-media_seg_name video0-$RepresentationID$-$Number$.mp4` : se especifica el nombre del resto de segmentos

`-utc_timing_url https://time.akamai.com/?iso` : se especifica un servidor de tiempo UTC para que el reproductor pueda sincronizar temporalmente el vídeo

`-remove_at_exit 0` : este valor define si se eliminan los archivos al terminar la grabación, en este caso está a cero indicando que no se deben borrar

`/var/www/html/segmentos/video.mpd` : por último se especifica la ruta donde se desea almacenar los segmentos, en este caso es una ruta del servidor web que los va a reproducir (más adelante se especifica qué solución se elige).

Los archivos que se generan son, un segmento inicial con el prefijo "init", un número de segmentos de contenido de vídeo y el archivo MPD. Cabe destacar que el segmento inicial no contiene vídeo en su interior, sino metadatos necesarios para la reproducción del vídeo. Esto es relevante a la hora de tratar y procesar los segmentos.

Esta va a ser la configuración elegida para realizar la emisión de vídeo. Ahora es momento de diseñar la manera de garantizar la integridad del vídeo.

6.4 Reproducción: Shaka Player sobre NGINX

A raíz de haber investigado los formatos y reproductores de vídeo adaptativo se ha decidido escoger Shaka Player. Esta decisión viene dada por el hecho de que nativamente soporta DASH, y se ha escogido DASH por ser compatible de manera más amplia que lo es HLS. Si se eligiese HLS se estaría limitando la compatibilidad de forma nativa a un colectivo de dispositivos vendidos por Apple. Este proyecto pretende ser compatible con la mayor cantidad de dispositivos que sea posible, y es por esto por lo que el formato DASH y Shaka Player son la combinación idónea.

A la hora de configurar un servicio de reproducción de vídeo, es imprescindible que exista un servidor web que almacene tanto la página web a mostrar como los fragmentos del vídeo. Tras haber considerado diferentes opciones, se escoge NGINX por la rapidez de funcionamiento, la sencillez del software, la cual ayuda a la fluidez del entorno y sobre todo por lo poco exigente que es en cuanto a consumo de recursos. El uso de una Rapsberry Pi marca y condiciona este tipo de decisiones ya que es un dispositivo moderadamente potente y requiere de optimización de recursos.

Una vez instalado el servidor web y puesto en marcha es posible desplegar nuestra web. Para poder utilizar Shaka Player de manera sencilla y sin tener que compilar e instalar todas las librerías, en este proyecto se escoge la opción de llamar a una CDN con las librerías compiladas. En este caso se utiliza la versión 3.0.1 de Shaka Player.

El HTML que va a interpretar el buscador utiliza los métodos de las librerías de Shaka:

```
<!DOCTYPE html>
  <html>
    <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/shaka-player/3.0.1/shaka-
player.compiled.js"></script>
    <script>
      var manifestUri = 'segmentos/video.mpd';
      function initApp() {
        shaka.polyfill.installAll();
        if (shaka.Player.isBrowserSupported()) {
          initPlayer();
        } else {
          console.error('Reproductor no soportado');
        }
      }
      function initPlayer() {
        var video = document.getElementById('video');
        var player = new shaka.Player(video);
        window.player = player;
        player.addEventListener('error', onErrorEvent);
        player.load(manifestUri).then(function() {
          console.log('Vídeo cargado correctamente');
        }).catch(onError);
      }
      function onErrorEvent(event) {
        onError(event.detail);
      }
      function onError(error) {
        console.error('Código de error: ', error.code, ' en ', error);
      }
      document.addEventListener('DOMContentLoaded', initApp);
    </script>
  </head>
  <body>
    <video id="video" width="1280" controls autoplay></video>
  </body>
</html>
```

Dentro del código se comprueba si el buscador es compatible y comienza el proceso de creación del reproductor.

Una vez se ha cargado la página web al completo (esto es posible controlarlo gracias al evento '*DomContentLoaded*') se inicia el reproductor.

El código tiene acceso al archivo *Media Presentation Description* (.mpd) y es capaz de buscar los segmentos que se encuentran en ese mismo directorio. En este caso es una carpeta llamada segmentos en un escalón por encima en jerarquía del servidor web.

Además, se especifica el ancho de la ventana, el cual está definido en 1280 píxeles, que como se ha comentado anteriormente, es la resolución elegida para este prototipo.

Tras haber configurado todos los servicios necesarios para realizar la emisión, es momento de investigar acerca de procesos de asegurar la integridad del contenido que se transmite.

Las emisiones de vídeo en directo tienen la capacidad de reproducirse en tiempo real, y además, en ocasiones, permiten ser reproducidas bajo demanda.

Este proyecto tiene como objetivo abordar ambos casos para definir qué método se ajusta mejor a cada uno. De este modo se pueden definir metodologías de procesado del vídeo diferentes para cada caso de uso.

6.5 Caso de uso 1: Verificación de la integridad para reproducción en tiempo real

Se podría decir que el caso de uso más común de consumir una emisión en directo es visualizándolo en tiempo real. Por ello, este primer caso de uso busca conseguir un enfoque que cubra la necesidad de mantener la información inalterada. Este enfoque debe conseguir que en el lado de la emisión se realice un procesado de los segmentos que pueda ser comprobado en destino. Por ello se divide el desarrollo en dos partes, procesado en origen y procesado de verificación en destino:

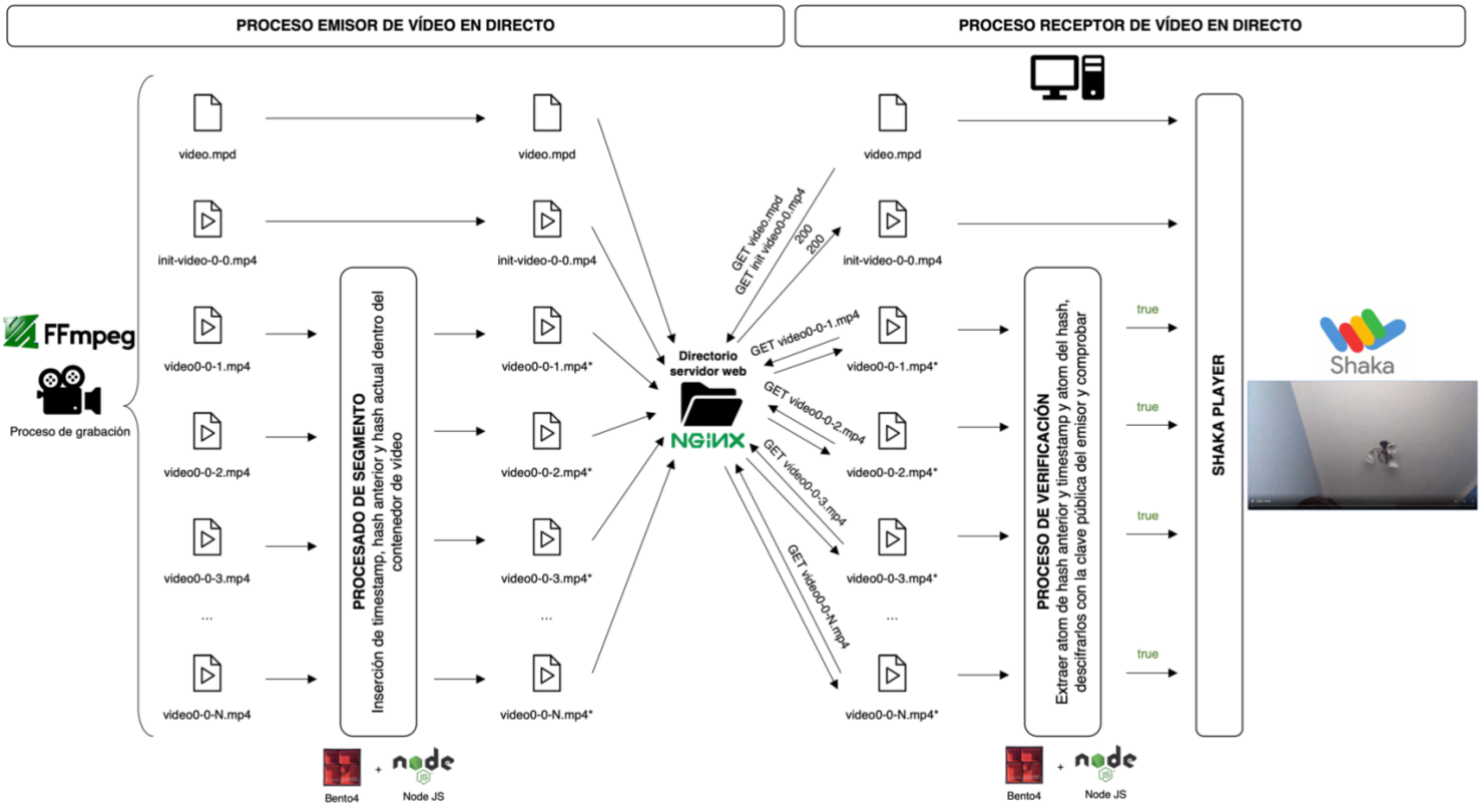


Fig. 12: Diagrama explicativo del primer caso de uso

La arquitectura propuesta consiste en procesar los segmentos grabados por FFmpeg dándoles forma de bloque (figura 2) y almacenarlos en un servidor web. Por otro lado, la parte interesada en consumir el vídeo en directo descarga los segmentos y verifica que la estructura interna de los bloques es correcta.

6.5.1 Procesado de segmentos en origen

Dado que la tecnología que se quiere aplicar es blockchain, la primera hipótesis planteada fue almacenar hashes de los diferentes archivos generados a la hora de emitir vídeo. Tras investigar el funcionamiento de la mayoría surgió el problema de que una blockchain tiene un tiempo de minado, que incluso en la blockchain con el tiempo de minado más bajo, es superior al tiempo de segmento que teníamos calculado utilizar. De acuerdo con lo investigado en el apartado 4.2.1, los segmentos no deben ser demasiado largos, por lo que aumentarlos haría que la eficiencia del sistema bajase.

Tras descartar el uso de una blockchain para almacenar dichos hashes, se puso a prueba una segunda hipótesis: insertar información relevante a la integridad del vídeo en un campo del *metadata* del vídeo. FFmpeg ofrece un comando que permite visualizar información del vídeo en profundidad:

```
ffmpeg -i /ruta/al/archive/videoN.mp4
```

Este comando devuelve error ya que no extrae del segmento toda la información que lo caracterizaría como un vídeo per se.

Analizando el tipo de archivo que genera FFmpeg se detectó que los segmentos generados no tenían una estructura de vídeo "completo". Es decir, los archivos de vídeo generados eran realmente vídeos fragmentados [21], lo que quiere decir que por sí solos no pueden tratarse como vídeos, sino que deben tratarse como conjunto. Por este motivo tampoco son reproducibles por un reproductor, ya que no contienen toda la información necesaria para poder leerlos.

De nuevo, tratando de encontrar información sobre una posible solución al problema, se encuentra otra posible solución. Al concatenar la información en binario del segmento inicial con la de el segmento enésimo, se genera un archivo de vídeo independiente. Este archivo ya es reproducible por reproductores de vídeo y además es legible para FFmpeg, por lo que el comando anterior mostraba la estructura del vídeo y del *metadata*. En este momento se decide insertar en una etiqueta del *metadata*, el hash del vídeo calculado a partir de la línea de comandos:

```
shasum -a 256 /ruta/al/archivo/videoN.mp4

ffmpeg -i /ruta/al/archivo/videoN.mp4 -movflags use_metadata_tags -metadata
hash=c55579c36aa545123dfa4e21e7fea414355c5c08342dae5ab66e11f300c0c5f8
videoN_conhash.mp4
```

Efectivamente, a la salida se obtuvo un archivo que, al inspeccionar su contenido, mostraba el valor de la etiqueta insertada. El problema que presentó este enfoque fue que desafortunadamente, FFmpeg no había respetado la estructura del vídeo sin la etiqueta, y había generado otro archivo completamente diferente. De hecho, al ejecutar el comando, se puede apreciar un procesado que dura un tiempo relativo a la duración del vídeo, por lo que FFmpeg transcodifica el vídeo. Esto desencadena en que el vídeo, a pesar de ser reproducible por reproductores de vídeo ya no es comprensible por Shaka Player, dado que la estructura interna que espera encontrar, no existe.

Gracias a la herramienta basada en MP4Box.js [22] y desarrollada en entorno web (con un front-end intuitivo) por parte de la escuela de ingenieros de telecomunicación de París, fue posible analizar los cambios entre ambos archivos:

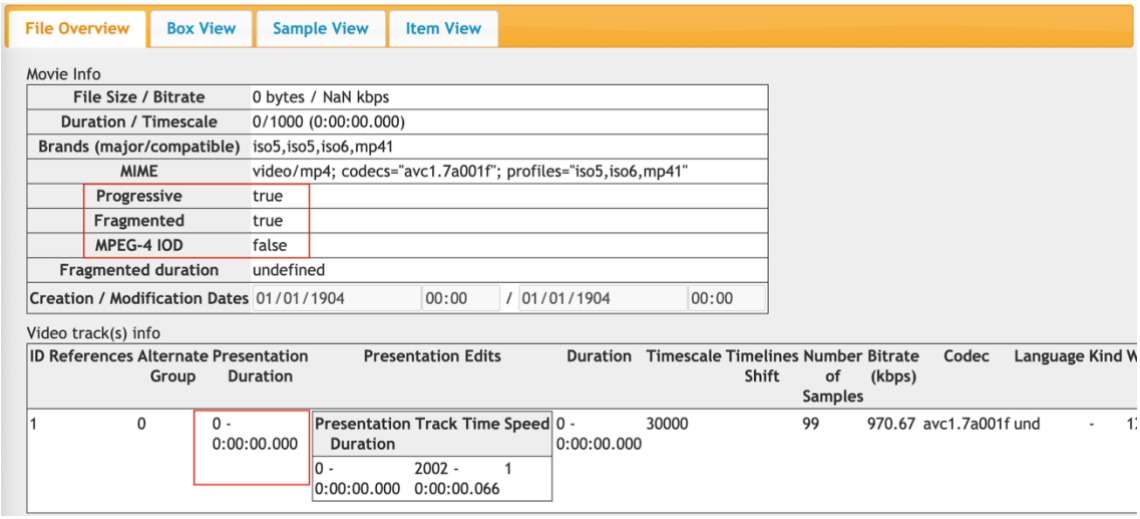


Fig. 13: Inspección de fragmento de vídeo generado por FFmpeg con formato DASH

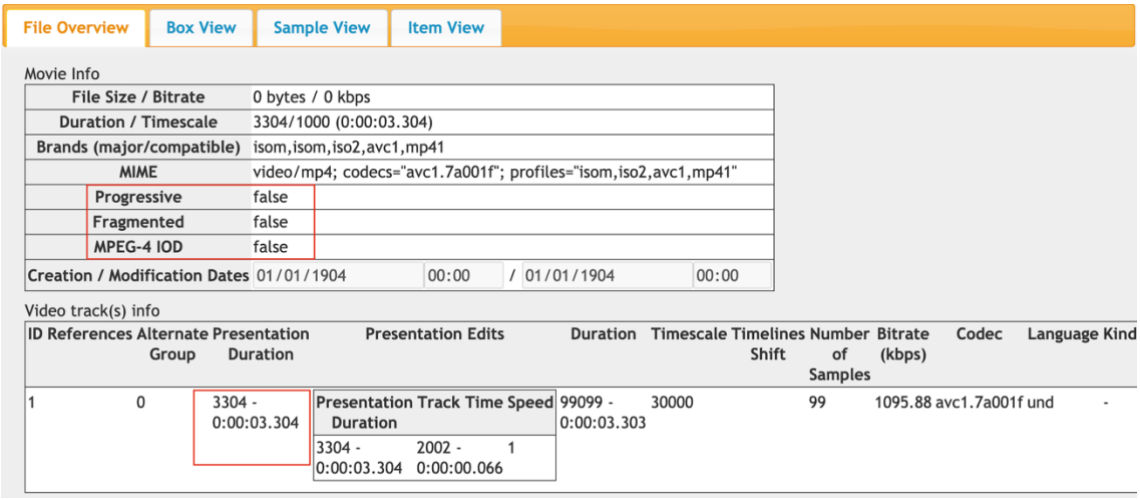


Fig. 14: Inspección de segmento DASH concatenado al segmento inicial

Tal y como se aprecia en las figuras 13 y 14, el vídeo pierde la característica de ser un vídeo fragmentado y de ser progresivo. Sumado a eso, la duración se ve alterada, ya que pasa de no tener información sobre la duración, a añadirse. Esto supuso el descarte de esta hipótesis de trabajo.

Como consecuencia de lo anterior, surgió la necesidad de analizar más en profundidad el contenido de un archivo fragmentado y en formato DASH. Herramientas como MP4Box.js [22] y Thumbcoil[23] hacen posible la inspección a nivel interno de archivos de vídeo:

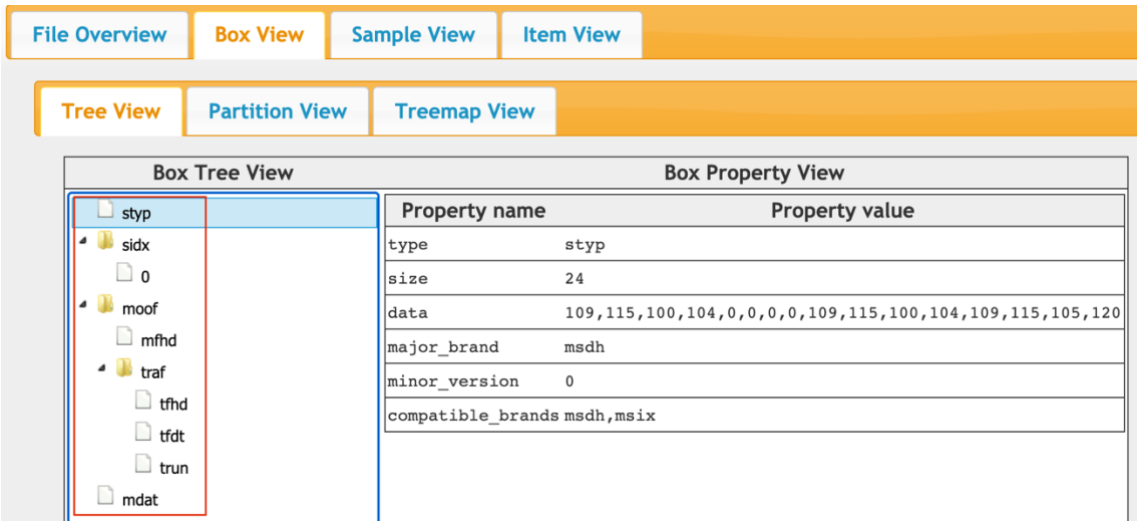


Fig. 15: Estructura del árbol de boxes o atoms de un segmento DASH

Como puede apreciarse en la figura 15, los contenedores mp4 de vídeo en formato DASH están compuestos por un ábol de atoms, que en el caso de ser generados por ffmpeg, incluyen la información que puede apreciarse en la imagen.

A partir de esta inspección surge la siguiente hipótesis, introducir información en forma de atom dentro del contenedor mp4.

Antes de insertar los atoms o boxes, es imprescindible conocer la estructura interna de los mismos, ya que de otro modo no se podrán insertar correctamente.

Tras obtener información acerca de herramientas que consiguen procesar archivos de vídeo a tan bajo nivel, surge la necesidad de utilizar Bento4. Este software nos ayudará a mostrar, insertar o extraer los atoms de un segmento para poder inspeccionarlos. Para poder hacer uso de él debemos acudir a la sección descargas de su página web, descargar los archivos fuente para todas las plataformas e instalarlos tal y como se detalla en el Anexo.

Con Bento4 preparado, resulta interesante hacer uso de la herramienta mp4dump que tiene Bento4, ya que analiza y muestra la estructura de árbol de atoms, su tamaño y cierta información:

```
./Bento4/Build/Targets/any-gnu-gcc/Debug/mp4dump /ruta/al/archivo/videoN.mp4
```

De este modo accedemos al directorio donde se almacenan los ejecutables de Bento4 y escogemos mp4extract, el cual se encarga de extraer atoms indicándole la ruta. Esta ejecución nos muestra por línea de comandos lo siguiente:

```
[styp] size=8+16
[sidx] size=12+40, version=1
  reference_ID = 1
  timescale = 15360
  earliest_presentation_time = 0
  first_offset = 0
[moof] size=8+1692
  [mfhd] size=12+4
    sequence number = 1
  [traf] size=8+1668
    [tfhd] size=12+16, flags=20038
      track ID = 1
      default sample duration = 512
      default sample size = 2126
      default sample flags = 1010000
    [tfdt] size=12+8, version=1
      base media decode time = 0
    [trun] size=12+1608, flags=601
      sample count = 200
      data offset = 1708
[mdat] size=8+835001
```

Se puede ver la estructura interna del segmento que le hemos indicado a Bento4 de manera más precisa. Para conocer mejor en qué consiste cada uno de los atoms se ha realizado una búsqueda del significado de los acrónimos de cada uno:

```
styp: Segment Type Box
sidx: Segment Index Box
moof: Movie Fragment Box
       mfhd:  Movie Fragment Header
       traf:  Track Fragment
              tfhd: Track Fragment Header
              tfdt: Track Fragment Decode Time
              trun: Track Fragment Run
mdat:  Media Data Container
```

Si se desea extraer el atom al completo para conseguir todavía mas información, gracias al comando `mp4extract` de Bento4, esto es posible:

```
./Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract styp
/ruta/al/archivo/videoN.mp4 /ruta/al/atom/styp.atom
```

En este caso, vamos a extraer el primer atom que aparece en la jerarquía. La ruta del atom está clara, tal y como se ve en la figura 16, el atom `styp` está en la parte más alta del árbol, por lo que no se encuentra dentro de ningún otro atom. Además, hay que indicar el vídeo a partir del cuál se quiere extraer y la ruta donde se quiere almacenar junto con el nombre del atom, que en este caso le hemos llamado con el mismo nombre.

Con el atom descargado y gracias a herramientas que analizan sintácticamente archivos (*parsing*) en formato hexadecimal mostrando una traducción a UTF8[25] podemos analizar el formato del atom extraído:

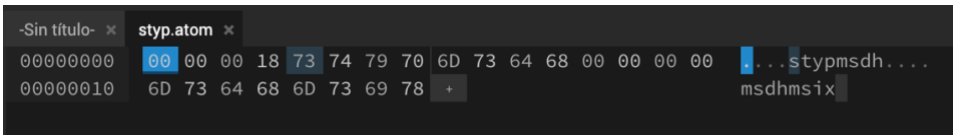


Fig. 16: estructura interna de un atom que forma parte de un contenedor mp4

Además, gracias a plantillas que se encuentran en foros especializados[26] en vídeo se puede entender mejor la estructura:

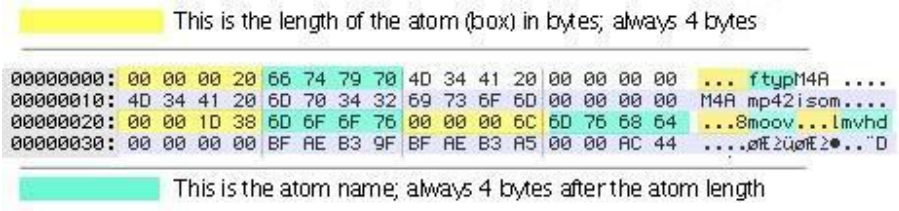


Fig. 17: Distribución de bytes dentro de un atom

Puede apreciarse que los cuatro primeros bytes indican el tamaño del atom en total, sumando ese mismo campo. Los cuatro siguientes indican el nombre del atom en formato hexadecimal, es decir, convirtiendo el carácter UTF8 a su codificación hexadecimal.

La siguiente hipótesis sería conseguir insertar dentro del contenedor del archivo, un atom escrito de manera personalizada, incluyendo la información que se considere relevante.

Para una primera prueba, se ha utilizado la librería File System de Node.js. Los detalles sobre su instalación se encuentran en el Anexo.

Gracias a este módulo, es posible generar archivos con el mismo formato que un atom del siguiente modo:

El primer paso es calcular cuánto ocupa la información que se desea almacenar, ya que de ello depende el primer campo del atom, ya que su estructura es:

```
4 bytes (tamaño) + 4 bytes (nombre) + N bytes (datos)
```

Para comenzar a escribir el archivo es necesario saber cuánto vale `4+4+N`.

Dado que en este proyecto se pretende almacenar un resumen hash SHA 256, N será igual a 32 bytes (256 bits / 8 bits/byte = 32 bytes). Por tanto, los primeros 4 bytes deberán almacenar el valor `4+4+32=40 bytes`, que en hexadecimal es `0x28`. Puesto que existen 4 bytes para representar el número, se introducirá: `00 00 00 28`.

A continuación, el campo del nombre del atom, en este caso no podemos insertar cualquier nombre de manera arbitraria. Esto se debe a que, en otra hipótesis de trabajo, se incluyó `hash` y `psha` como nombres de atom (traducido a hexadecimal[27]: `hash = 68 61 73 68, psha = 70 73 68 61`). Tras realizar diferentes pruebas, se comprobó que modificar el árbol de atoms de un archivo de vídeo normal no suponía ningún problema a la hora de reproducirlo, pero no ocurrió lo mismo al alterar la estructura de un segmento en formato DASH.

Este enfoque finalizó con Shaka Player no reconociendo el archivo y provocando un error de análisis sintáctico del *MediaSource Extension* del buscador (Google Chrome): "*BoxReader::IsValidTopLevelBox() issues a MEDIA_LOG if an unrecognized top-level box is found in the path that returns false, and typically results in parse error*". Gracias al equipo de soporte de Shaka Player[29] y Chrome[28], se pudo conocer que ese bug se podía evitar sin introducir nombres arbitrarios de atoms.

Finalmente, tras consultar un extracto del estándar ISO 14496 donde se detalla la estructura de atoms, se obtuvo una visión más clara de la estructura de un contenedor mp4 y se encontró la solución respaldada por el estándar:

moof							8.8.4	*movie fragment*
	mfhd					*	8.8.5	*movie fragment header*
	meta						8.11.1	*metadata*
	traf						8.8.6	*track fragment*
		tfhd				*	8.8.7	*track fragment header*
		trun					8.8.8	*track fragment run*
		sbgp					8.9.2	*sample-to-group*
		sgpd					8.9.3	*sample group description*
		subs					8.7.7	*sub-sample information*
		saiz					8.7.8	*sample auxiliary information sizes*
		saio					8.7.9	*sample auxiliary information offsets*
		tfdt					8.8.12	*track fragment decode time*
		meta					8.11.1	*metadata*
mfra							8.8.9	*movie fragment random access*
	tfra						8.8.10	*track fragment random access*
	mfro					*	8.8.11	*movie fragment random access offset*
mdat							8.2.2	*media data container*
free							8.1.2	*free space*
skip							8.1.2	*free space*
	udta						8.10.1	*user-data*
		cprt					8.10.2	*copyright etc.*
		tsel					8.10.3	*track selection box*
		strk					8.14.3	*sub track box*
			stri				8.14.4	*sub track information box*
			strd				8.14.5	*sub track definition box*
meta							8.11.1	*metadata*

Fig. 18: Extracto de la lista de atoms que se refleja en el estándar ISO 14496

Puede apreciarse que los atoms `free` y `skip` permiten al usuario introducir información, ya que se clasifican como contenedores de espacio libre.

Tomando como referencia el funcionamiento de Naivechain citado en soluciones existentes, se diseña un método de trabajo para conseguir la solución buscada de la siguiente manera:

1) Introducir dentro del atom `free` un timestamp del momento de la generación de dicho segmento y el hash del segmento anterior. En caso de ser el primer segmento, se le introduce el hash del segmento inicial, que como se ha comentado en el apartado de grabación, se tratará de manera diferente debido a que no contiene información de vídeo como tal, sino más bien metadatos necesarios para la reproducción.
2) Una vez se ha introducido la información de timestamp y del hash del segmento anterior, se calcula el hash del segmento con el atom `free` insertado.
3) La información del hash calculado en 2) se inserta en el atom `skip` para ser introducido al segmento.

Tras poner a prueba el mecanismo anterior, se comprueba que efectivamente, los segmentos se generan de manera correcta y siguen siendo reproducibles por Shaka Player. Por lo que es el enfoque definitivo en este caso de uso. Lo siguiente será automatizar el proceso para que se lleve a cabo lo más próximo a tiempo real que sea posible. Como se ha comentado anteriormente, la herramienta elegida para la automatización será Node.js.

Para el desarrollo de la solución, además de la librería `file-system` (que ayuda a interactuar con archivos permitiendo su lectura y escritura) se necesita la librería `crypto` (imprescindible para calcular hashes) y `child_process` (permite ejecutar código desde la línea de comandos). Los detalles sobre su instalación se detallan en el Anexo.

Tras haber instalado los módulos, el desarrollo de la solución se ha diseñado del siguiente modo:

```
1  const fs = require('fs');
2  const crypto = require('crypto');
3  const exec = require('child_process');
4
5  var prevhash ='';
6  var hash='';
7  var blockHash;
8  var fragNum = 1;
9  var index = 0; //chain index
10 var Frag='';
11 var limit=0;
12
13 checkFlag();
14
15 function checkFlag(){
16
17     if (fragNum>1){
18     prevhash=blockHash;
19
20     }
21     else{
22     var hashinit = fs.readFileSync('/var/www/html/segmentos/init-video0-0.mp4', 'binary').toString();
23     prevhash = crypto.createHash('sha256').update(hashinit).digest('hex');
24     }
25
26     Frag='video0-0-'+fragNum.toString();
27
28     var flag = fs.existsSync('/var/www/html/segmentos/'+Frag+'.mp4');
29
30     if (flag == false){
31         if(limit < 150){
32         limit++;
33         setTimeout(checkFlag, 100);
34         }
35         else{
36
37         }
38     } else {
39         limit=0;
40         calcHash();
41     }
42 }
```

Fig. 19: Código de automatización de inserción de hashes dentro del contenedor de vídeo DASH mp4 (1/4)

Las primeras tres líneas se encargan de llamar a los módulos requeridos por el proceso. A continuación de inicializan las variables a utilizar y se llama a la primera función del proceso.

Esta primera función se encarga de asignar un valor de hash anterior al segmento que se lee con la variable `prevhash`. Tal y como se ha comentado anteriormente, en caso de ser el primer segmento el que se ha de tratar, se le asigna el hash del segmento inicial.

A continuación, se genera un nombre de fragmento que va a ir variando en función del valor de una variable (se incrementará de manera secuencial). A partir del nombre del segmento se hace uso de la librería `file-system` para comprobar la existencia del fragmento. La ubicación es la que utiliza FFmpeg para generar los segmentos.

Al iniciar la grabación los primeros archivos que se forman son el MPD y el inicial de manera temporal. Instantes más tarde se genera el primer segmento con contenido de vídeo (`video0-0-1.mp4` en este caso) y el segmento inicial se actualiza para quedar en su estado definitivo. Es decir, el segmento inicial depende de la formación del primero con composición de vídeo, es por ello que, se espera a que exista `video0-0-1.mp4` y no al segmento inicial. Es un indicativo de que ambos ya están listos para ser procesados.

Si existe `video0-0-1.mp4` se pasa a la siguiente función. Si no existe en el momento de la comprobación, se ha implementado un temporizador que da un margen para que se inicie la grabación y será el mismo indicador que nos avisará de si la grabación ha finalizado. En este caso se comprueba cada 100 ms la existencia del archivo y se espera hasta un total de 150 iteraciones (15 segundos) para abortar la ejecución. La variable `limit` donde se almacena dicho número de iteraciones, se reinicia al encontrar el segmento (línea 39) para darle el mismo margen al siguiente y así sucesivamente.

```
45  function calcHash(){
46      //hash_N = sha256(index, hash_N-1, timestamp, segmento_N)
47
48      var skip = '0000003B736B6970';
49
50      var timestamp = new Date().toISOString().replace(/T/, ' ').replace(/\..+/, '').toString();
51      var wtimestamp = new Buffer.from(timestamp, 'binary').toString('hex');
52      console.log(wtimestamp);
53      var prevAtom = skip+prevhash+wtimestamp;
54
55      var buffer = new Buffer.from(prevAtom, 'hex');
56
57      fs.writeFileSync('/var/www/html/segmentos/prev.atom', buffer, 'binary');
58
59      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract styp /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/styp.atom');
60      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract sidx /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/sidx.atom');
61      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract moof /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/moof.atom');
62      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract mdat /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/mdat.atom');
63
64      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4edit --insert :/var/www/html/segmentos/styp.atom --insert :/var/www/html/segmentos/
            sidx.atom --insert :/var/www/html/segmentos/moof.atom --insert :/var/www/html/segmentos/mdat.atom --insert :/var/www/html/segmentos/prev.atom /var/www/html/
            segmentos/'+Frag+'.mp4 /var/www/html/segmentos/'+Frag+'.mp4');
65
66      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract skip /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/prevext.atom');
67
68      var prevHashIn=fs.readFileSync('/var/www/html/segmentos/prevext.atom', 'hex').toString().slice(16, 80);
69      var rtimestamp=fs.readFileSync('/var/www/html/segmentos/prevext.atom', 'hex').toString().slice(80);
70      var stimestamp=new Buffer.from(rtimestamp, 'hex').toString('binary');
71
72      console.log(index);
73
74      var segBinario = fs.readFileSync('/var/www/html/segmentos/'+Frag+'.mp4', 'binary').toString();
75
76      console.log('indice: '+index);
77      console.log('timestamp: '+stimestamp);
78      console.log(Frag+' prev: '+prevHashIn);
79
```

Fig. 20: Código de automatización de inserción de hashes dentro del contenedor de vídeo DASH mp4 (2/4)

Una vez se ha detectado la existencia del fragmento que se quiere tratar, es momento de generar el atom para insertarlo en el contenedor mp4. Siguiendo el procedimiento explicado anteriormente, para introducir el atom `skip` debemos introducir el tamaño del atom y la palabra `skip` en hexadecimal.

Ya se conoce que el hash SHA 256 ocupa 32 bytes. Es momento de calcular el tamaño del timestamp. En este caso va a ser una fecha ISO UTC (línea 50) con formato AAAA-MM-DD HH:MM:SS. Contando espacios, en total son 19 caracteres, que se codifican en 19 bytes hexadecimales (línea 51). Se puede calcular el total del tamaño del atom a partir de estos datos.

En este caso tendrá un tamaño de: 4 bytes de cabecera + 4 bytes de nombre + 59 bytes (32 bytes del hash anterior + 19 bytes del timestamp).

Para que el atom tenga el formato adecuado se debe guardar la información a partir de un buffer que la codifica en formato hexadecimal, es decir, igual que el que genera FFmpeg en la emisión. Ese buffer se utiliza para que la herramienta `file-system` genere el archivo que vamos a insertar. Para este primer atom, se le asocia un nombre que resulte más intuitivo que `skip`, y es `prev` indicativo de que contiene el hash previo.

A partir de este punto del código, desde la línea 59 hasta la 62 se realiza la extracción de todos los atoms del segmento. El motivo es que, la herramienta Bento4 dispone de la capacidad de insertar un atom de manera aislada, pero al parecer está preparada para procesar archivos de vídeo "independientes" y no fragmentados. Al intentar insertar un atom a un segmento DASH la herramienta tiene un comportamiento erróneo eliminando todos los atoms que existían anteriormente (`styp`, `sidx`, `moof` y `mdat`) e insertando únicamente el "nuevo". Por ello, la solución ha sido extraer todos los atoms e insertarlos juntos en la línea 64 con el comando `mp4edit --insert` donde se le indica la ruta de todos los atoms que deben ser insertados.

De las líneas 66 a la 70 se lleva a cabo una comprobación de que el hash se ha insertado correctamente. Se utiliza la herramienta `mp4extract` de nuevo y se leen los bytes donde se ha

almacenado cada dato (para hacer un `console.log` y comprobar el valor). Como aclaración, la herramienta `mp4extract` copia el atom del archivo de vídeo pero no lo elimina, por lo que tras la comprobación no es necesario insertarlo de nuevo.

En este punto, se ha completado el primer paso del procedimiento que se explica al definir el enfoque definitivo de la solución. El siguiente paso es calcular el hash del segmento con el atom `skip` insertado. Para ello en la línea 74 se hace uso de la herramienta file-system y se lee el contenido del segmento en formato binario, es decir, el contenido de los atoms `styp`, `sidx`, `moof`, `mdat` y `skip`.

```
80      var hashString = segBinario;
81
82      blockHash = crypto.createHash('sha256').update(hashString).digest('hex');
83
84      var buffer = new Buffer.from(blockHash, 'binary');
85      console.log(Frag+' hash: '+blockHash);
86
87      var free = '0000002866726565';
88      var hashAtom = free+blockHash;
89
90      var buffer = new Buffer.from(hashAtom, 'hex');
91      fs.writeFileSync('/var/www/html/segmentos/hash.atom', buffer, 'binary');
92
93      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract styp /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/styp.atom');
94      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract sidx /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/sidx.atom');
95      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract moof /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/moof.atom');
96      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract mdat /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/mdat.atom');
97      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract skip /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/prev.atom');
98
99      exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4edit --insert :/var/www/html/segmentos/styp.atom --insert :/var/www/html/segmentos/
            sidx.atom --insert :/var/www/html/segmentos/moof.atom --insert :/var/www/html/segmentos/mdat.atom --insert :/var/www/html/segmentos/prev.atom --insert :/var
            /www/html/segmentos/hash.atom /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/'+Frag+'.mp4');
100
101     exec.execSync('sudo /usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract free /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/segmentos/hashext.atom');
102
103     var hashIn=fs.readFileSync('/var/www/html/segmentos/hashext.atom', 'hex').toString().slice(16);
104
105     nextFrag(Frag);
106
107 }
```

Fig. 21: Código de automatización de inserción de hashes dentro del contenedor de vídeo DASH mp4 (3/4)

Se calcula el hash del segmento haciendo uso de la librería `crypto` (línea 82) y se almacena en la variable `blockHash`. El nombre se debe a que, de manera análoga a una blockchain, este bloque va a contener un timestamp, el hash anterior, datos (el propio segmento) y además un resumen de todo lo anterior. Todo ello en un mismo archivo con extensión mp4.

Para finalizar el proceso se debe volver a calcular el tamaño del segundo atom a insertar, que va a tener una estructura: 4 bytes (tamaño) + 4 bytes (nombre) + 32 bytes (hash del "bloque"). Esto es 40 bytes, que en hexadecimal queda `0x28`. Se puede ver en la línea 87 cómo se genera una variable que contiene los campos de tamaño y de nombre, donde `00 00 00 28` son los 40 bytes y `66 72 65 65` es la palabra free en hexadecimal.

Toda la información unida (línea 88) se guarda con el mismo método que el atom anterior (líneas 90, 91) y se sigue un procedimiento análogo de extracción de todos los atoms (esta vez también el `skip`) y se vuelven a insertar junto con el último, `free`.

Se comprueba de nuevo que la información insertada es correcta (línea 103) y se continúa la ejecución con el siguiente paso:

```
110 function nextFrag(){
111
112     var flag = fs.existsSync('/var/www/html/segmentos/hashext.atom');
113     exec.execSync('rm /var/www/html/segmentos/*.atom');
114
115     if (flag == true){
116         if (fragNum>1){
117             exec.execSync('cp -rf /var/www/html/segmentos/video.mpd /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/reproductor');
118         }
119         else{
120             exec.execSync('cp /var/www/html/segmentos/video.mpd /var/www/html/segmentos/init-video0-0.mp4 /var/www/html/segmentos/'+Frag+'.mp4 /var/www/html/
                reproductor');
121         }
122
123     fragNum++;
124     index++;
125
126     checkFlag();
127
128     } else {
129         //FIN
130
131     }
132 }
```

Fig. 22: Código de automatización de inserción de hashes dentro del contenedor de vídeo DASH mp4 (4/4)

La última sección del código se encarga de eliminar todos los archivos con extensión `.atom` del directorio antes de comenzar con el procesado del siguiente segmento (línea 113).

Además, se comprueba que el atom extraído en el paso anterior existe como prueba de que el procesado ha finalizado (líneas 112 y de la 115 hasta el final). En caso de que exista se comprueba qué fragmento es el que se está procesando, ya que en caso de ser el primero, se debe mover al directorio del servidor web: el segmento preparado para su reproducción y descarga, el segmento inicial y el MPD (línea 120). A partir de ese momento, el resto de segmentos se tratan por igual (línea 117) y se mueve el segmento y el archivo MPD (el cual cambia con cada segmento que se genera, por lo que es necesario actualizarlo en el servidor siempre a su última versión) al directorio del servidor web. La carpeta donde se mueven se llama `reproductor` y será donde se deba acceder para descargar los segmentos en destino.

Después, se actualiza el número de fragmento y el "índice" del bloque (utilizado para mostrar el progreso del proceso en la línea de comandos). Una vez se ha actualizado el número de fragmento se vuelve a llamar a la función inicial que espera a que exista el siguiente fragmento y así hasta el fin de la emisión.

Este proceso está preparado para ejecutarse en paralelo a la grabación o emisión de vídeo y está optimizado para tratar los segmentos lo más cercano a tiempo real como sea posible. De hecho, durante su funcionamiento no se aprecia retraso debido al procesado a tan bajo nivel que se lleva a cabo.

6.5.2 Verificación de segmentos en destino

Tras haber conseguido un método eficiente de procesar los segmentos y transportar información relevante a la integridad del vídeo es momento de desarrollar el proceso que se lleva a cabo a la hora de consumir ese contenido, que al igual que el procesado anterior, debe estar guiado bajo la premisa de poder realizarse lo más cercano a tiempo real como sea posible.

La solución planteada funciona, al igual que el proceso de emisión, usando Node.js sobre JavaScript, basándose en la arquitectura propuesta en la figura 14.

```
1  const http = require('http');
2  var exec = require('child_process');
3  const fs = require('fs');
4  const crypto = require('crypto');
5
6  var Frag='';
7  var i=1; //frag index
8  var index=0; //block index
9  var file;
10 var flag2;
11 var flag3;
12 var flag4;
13 var calcHash='';
14 var prevhash;
15 var Frag2;
16 var limit=0;
17
18 checkUrl();
19
20 function checkUrl(){
21
22     Frag = 'video0-0-'+i;
23     var url = 'http://localhost:80/reproductor/'+Frag+'.mp4';
24
25     http.get(url, (res)=>{
26         var {statusCode} = res;
27         if(statusCode==200){
28             limit=0;
29             i++;
30             downloadFiles(Frag);
31             }
32         else{
33             if (limit<150){
34                 limit++;
35                 setTimeout(checkUrl, 100);
36             }
37             else{
38             }
39         }
40     }).on('error', (e)=>{
41         console.log('ERROR. El archivo '+Frag+'.mp4 no existe.');
42     });
43
44 }
```

Fig. 23: Código de automatización de extracción y verificación de hashes de vídeo DASH mp4 (1/4)

Se pueden apreciar similitudes con respecto al programa anterior debido a que se realiza el proceso inverso. En las primeras cuatro líneas encontramos los módulos `child_process`, `file-system` y `crypto` igual que en el programa anterior con la diferencia de la inclusión del módulo `http`, el cual nos va a permitir realizar una petición al servidor web que aloja los archivos y poder descargar los segmentos. Los detalles sobre la instalación se encuentran en el Anexo.

Entre las líneas 6 y 16 se declaran variables fuera de funciones y condiciones para que poder actualizar su valor dentro de estas secciones de código de manera global.

El mecanismo de comprobación de existencia de un fragmento es el mismo exactamente que en el apartado anterior exceptuando que ahora no es `file-system` quien lee un archivo almacenado en local, sino el módulo `http` quien realiza una petición GET al servidor web para conseguir una respuesta que nos indique si existe (respuesta código 200 HTTP) o no existe (respuesta código 404 HTTP). Dispone de un temporizador, igual que en el código anterior y en caso de ser sobrepasado, se aborta la ejecución (línea 37, `else` vacío).

En el caso de que el fragmento se encuentre en el servidor web, se procede a continuar la ejecución del código:

```
46  async function downloadFiles(){
47
48      if(i==2){
49
50              var flag1=false;
51              file = fs.createWriteStream('/var/www/html/test/init-video0-0.mp4');
52              var request = http.get('http://localhost:80/reproductor/init-video0-0.mp4', function(response){response.pipe(file);});
53              var endfile = new Promise(function(resolve, reject){
54                  file.on('close',()=> resolve(flag1=true));
55              })
56              let filef1 = await endfile;
57      }
58
59      flag2=false;
60      flag3=false;
61
62      file = fs.createWriteStream('/var/www/html/test/video.mpd');
63      var request = http.get('http://localhost:80/reproductor/video.mpd', function(response){response.pipe(file);});
64      var endfile = new Promise(function(resolve, reject){
65          file.on('close',()=> resolve(flag3=true));
66      })
67      let filef0 = await endfile;
68
69      file = fs.createWriteStream('/var/www/html/test/'+Frag+'.mp4');
70      var request = http.get('http://localhost:80/reproductor/'+Frag+'.mp4', function(response){response.pipe(file);});
71      var endfile = new Promise(function(resolve, reject){
72          file.on('close',()=> resolve(flag2=true));
73      })
74      let filef = await endfile;
75
76
77      if ((flag2&&flag3)==true){
78      console.log('Tamaño de '+Frag+'.mp4 (bytes): '+fs.statSync('/var/www/html/test/'+Frag+'.mp4')['size']);
79      console.log('');
80      segName();
81      }
82
83  }
```

Fig. 24: Código de automatización de extracción y verificación de hashes de vídeo DASH mp4 (2/4)

Dado que la variable `i` se declara con un valor igual a 1 para comenzar a buscar el segmento multimedia número 1 y que se actualiza en caso de éxito antes de pasar a la siguiente función (líneas 29 y 30 de la figura 24), para filtrar el primer momento en que se llama a la función `downloadFiles` hay que filtrar cuándo la variable i es igual a 2 (preparada para la siguiente iteración, pero realmente estando todavía en la primera iteración).

En esa primera iteración solamente se va a entrar a la sentencia `if` y se va a descargar el segmento inicial, ya que sólo queremos que se descargue una vez. En el resto de iteraciones se omitirá ese paso y se seguirá la ejecución desde la línea 59.

Es en esta línea y la siguiente donde se declaran dos *flags*, que se utilizarán como indicadores de que la descarga ha finalizado de los dos siguientes archivos.

Como se ha comentado en el apartado anterior, el archivo MPD varía durante la emisión del vídeo, por lo que se hace imprescindible descargar continuamente este archivo para que la emisión se realice correctamente. Además, se descarga el segmento multimedia correspondiente a dicha iteración.

La librería `file-system` hace posible que escribamos un flujo de datos sobre un archivo a raíz de una petición `get` hecha gracias al módulo `http` (líneas 50 a 56, 62 a 67 y 69 a 74). Además, esta función del código es de tipo asíncrona, por lo que hay que esperar a que los archivos se hayan descargado con el comando `await`.

Una vez se ha finalizado la descarga de los archivos, los `flags` que nombrábamos anteriormente cambian de estado `false` a `true` y se continúa la ejecución del código. Además, como método de seguimiento del proceso se muestra el tamaño del segmento descargado.

El directorio escogido para esta prueba es diferente al directorio donde la cámara deposita la información (apartado 6.5.1 de la memoria). Así se pretende que la descarga desde el servidor

web ponga a prueba la infraestructura generada con NGINX simulando un caso de descarga en remoto.

La ejecución, tras la descarga de los archivos continúa del siguiente modo:

```
86  function segName(){
87
88      if(index>0){
89          prevhash = calcHash;
90
91      }
92      else{
93          var hashinit = fs.readFileSync('/var/www/html/test/init-video0-0.mp4', 'binary').toString();
94          prevhash = crypto.createHash('sha256').update(hashinit).digest('hex');
95      }
96
97      Frag2 = 'video0-0-'+(index+1).toString();
98
99      // console.log(timeStamp);
100
101     exec.execSync('/usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract styp /var/www/html/test/'+Frag2+'.mp4 /var/www/html/test/styp.atom');
102     exec.execSync('/usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract sidx /var/www/html/test/'+Frag2+'.mp4 /var/www/html/test/sidx.atom');
103     exec.execSync('/usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract moof /var/www/html/test/'+Frag2+'.mp4 /var/www/html/test/moof.atom');
104     exec.execSync('/usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract mdat /var/www/html/test/'+Frag2+'.mp4 /var/www/html/test/mdat.atom');
105     exec.execSync('/usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract skip /var/www/html/test/'+Frag2+'.mp4 /var/www/html/test/prev.atom');
106     exec.execSync('/usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4extract free /var/www/html/test/'+Frag2+'.mp4 /var/www/html/test/hash.atom');
107
108     exec.execSync('/usr/src/Bento4/Build/Targets/any-gnu-gcc/Debug/mp4edit --insert :/var/www/html/test/styp.atom --insert :/var/www/html/test/sidx.atom --insert
        :/var/www/html/test/moof.atom --insert :/var/www/html/test/mdat.atom --insert :/var/www/html/test/prev.atom /var/www/html/test/'+Frag2+'.mp4 /var/www/html/
        test/'+Frag2+'.mp4');
109
110     var segBinario = fs.readFileSync('/var/www/html/test/'+Frag2+'.mp4', 'binary').toString();
111
112     var rprevhash = fs.readFileSync('/var/www/html/test/prev.atom', 'hex').toString().slice(16, 80);
113
114     var rtimestamp = fs.readFileSync('/var/www/html/test/prev.atom', 'hex').toString().slice(80);
115     var stimestamp = new Buffer.from(rtimestamp, 'hex').toString('binary');
116
117     //console.log(rprevhash);
118     //console.log(stimestamp);
```

Fig. 25: Código de automatización de extracción y verificación de hashes de vídeo DASH mp4 (3/4)

La siguiente porción de código es muy similar a la vista en el apartado anterior (6.5.1) ya que comienza actualizando un valor de hash anterior y generando el nombre del fragmento a tratar a partir de un índice.

En este momento lo que se pretende es extraer el atom que contiene el hash del "bloque", es decir, el último que fue insertado en el apartado anterior, y analizar s contenido.

Como se ha explicado anteriormente, Bento4 expulsa todos los atoms al intentar modificar la estructura interna de un segmento DASH (comportamiento erróneo), por lo que se va a seguir el mismo procedimiento que antes de: extraer todos los atoms (líneas 101 a 106) e insertar todos menos el que contiene el hash del bloque (línea 108), que en este caso está alojado en el atom `free`.

Una vez se tiene el segmento sin ese atom, se lee su contenido en binario y se almacena (línea 110), este es el hash que se ha descargado y se debe comprobar que coincide con el calculado a partir del resto de partes del contenedor mp4.

Tras esto, se lee el atom `skip` que contiene información del timestamp y del hash anterior. Concretamente se lee entre sus posiciones 16 y 80 para obtener el hash anterior del bloque y a partir de la 80 para leer el timestamp (líneas 112 y 114). Esto se realiza para comprobar que la información se ha almacenado de manera correcta.

La ejecución a partir de este punto continúa:

```
120     var hashString = segBinario;
121
122     calcHash = crypto.createHash('sha256').update(hashString).digest('hex');
123     console.log('Hash calculado de '+Frag2+'.mp4: '+calcHash);
124
125
126     var blockHash = fs.readFileSync('/var/www/html/test/hash.atom', 'hex').toString().slice(16);
127     console.log('Hash descargado de '+Frag2+'.mp4: '+blockHash);
128
129     exec.execSync('rm /var/www/html/test/*.atom');
130
131     if (blockHash == calcHash){
132         console.log('Exito. El hash descargado y el calculado para el segmento '+Frag2+'.mp4 coinciden.\n');
133     }
134     else{
135         console.log('Error. El hash descargado no coincide con el calculado para el segmento '+Frag2+'.mp4.\n');
136     }
137
138
139     index++;
140
141     file.end();
142     file.close();
143
144     checkUrl();
145
146 }
```

Fig. 26: Código de automatización de extracción y verificación de hashes de vídeo DASH mp4 (4/4)

Se calcula el hash del segmento sin el atom `free` (el cual contiene el valor del hash del bloque) para comprobar que coincide con el valor que existe dentro del atom, que ya ha sido extraído (líneas 122 y 123 se calcula el hash y línea 126 se lee el contenido del atom).

Si coinciden significa que ningún elemento del vídeo ha sido alterado y se mostrará por consola un mensaje comunicándolo (línea 132). En caso contrario, si no coinciden, se mostrará un mensaje de error (línea 135).

Además, para "limpiar" el directorio antes de la siguiente iteración, se eliminan todos los archivos de extensión `.atom` (línea 129).

Por último, se actualiza el índice del segmento y se vuelve a ejecutar la rutina.

Gracias a este método, el receptor de los archivos es capaz de realizar una verificación en directo de que los fragmentos que recibe son correctos, asegurando la integridad del vídeo.

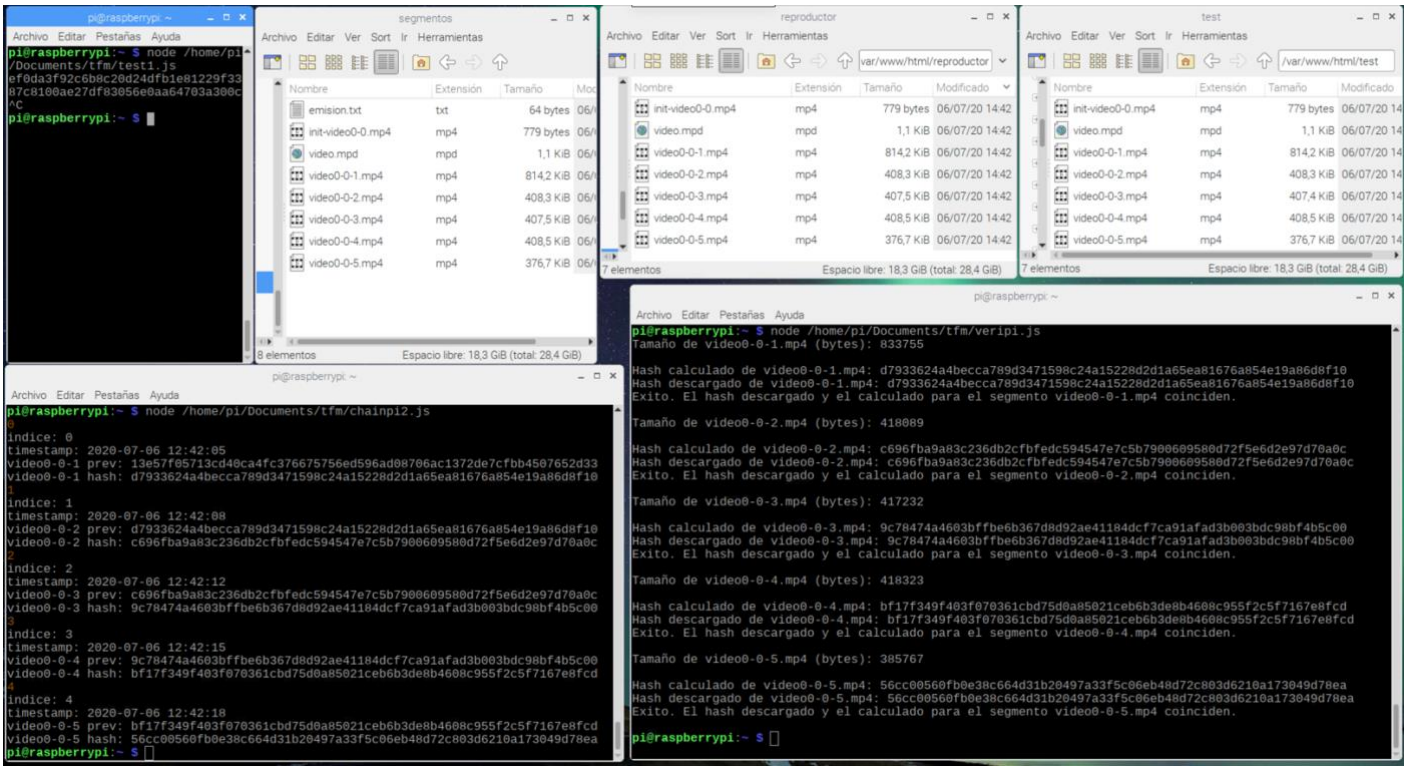


Fig. 27: Resultado de la ejecución tanto en origen como en destino del procesado basado en blockchain

En la figura 27 se puede ver el proceso de grabación en la parte superior izquierda, el cual muestra el identificador de la emisión. Este proceso deposita los segmentos en la ventana llamada segmentos de la figura.

La ventana de comandos situada en la parte inferior izquierda es la encargada de procesar los segmentos generados por FFmpeg y procesarlos, para más tarde copiarlos al directorio del servidor web, llamado reproductor.

Por último, la ventana de comandos situada en la parte inferior derecha se encarga de simular el proceso de descarga por parte del interesado en consumir la emisión de vídeo, de manera que descarga y procesa los segmentos, verificando que su estructura interna es correcta.

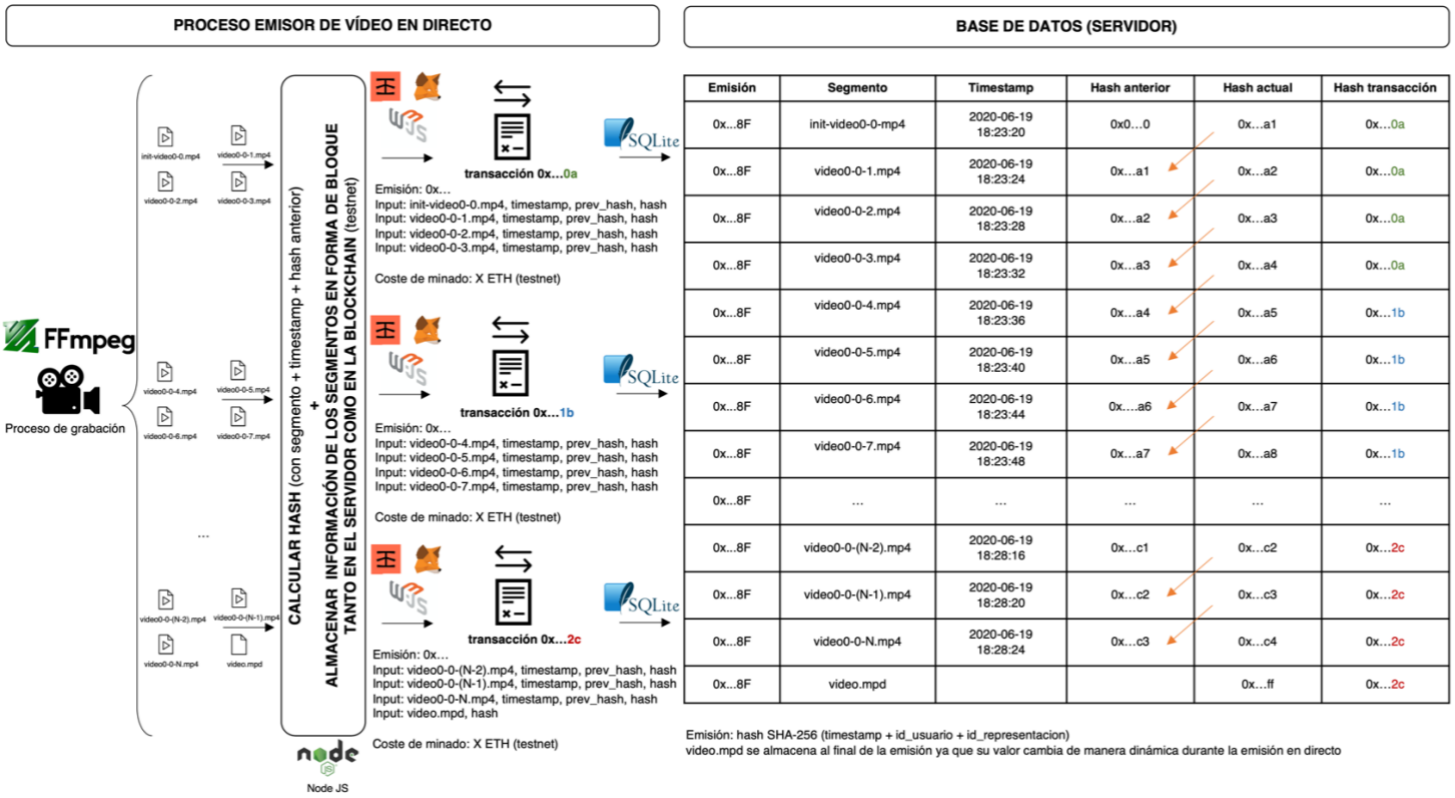
En ambas ventanas inferiores se pueden ver los mensajes de seguimiento del proceso. A la hora de desarrollar la herramienta suponen una fuente de información muy importante, al conocer en detalle qué ocurre en cada caso.

6.6 Caso de uso 2: Verificación de la integridad para reproducción bajo demanda

El presente caso de uso va a centrarse en implementar una solución que vaya enfocada a la verificación de los segmentos consumidos bajo demanda. Es decir, a diferencia del anterior caso de uso, se va a sacrificar una diferencia de tiempo ligeramente mayor entre el tiempo de emisión y el de registro en la blockchain, sin perder de vista el objetivo de que se acerque a tratamiento en tiempo real tanto como sea posible. Además, el valor añadido de esta solución es la posibilidad de asegurar la integridad del vídeo de manera inmutable.

6.6.1 Emisión y registro de los segmentos de la emisión en la blockchain

Esta solución busca realizar una transacción en una blockchain de pruebas mediante el despliegue de un Smart Contract que almacene los valores que se determinen importantes en la red, pagando la transacción utilizando fondos asociados a una dirección de una cartera.



Emisión	Segmento	Timestamp	Hash anterior	Hash actual	Hash transacción
0x...8F	init-video0-0-mp4	2020-06-19 18:23:20	0x0...0	0x...a1	0x...0a
0x...8F	video0-0-1.mp4	2020-06-19 18:23:24	0x...a1	0x...a2	0x...0a
0x...8F	video0-0-2.mp4	2020-06-19 18:23:28	0x...a2	0x...a3	0x...0a
0x...8F	video0-0-3.mp4	2020-06-19 18:23:32	0x...a3	0x...a4	0x...0a
0x...8F	video0-0-4.mp4	2020-06-19 18:23:36	0x...a4	0x...a5	0x...1b
0x...8F	video0-0-5.mp4	2020-06-19 18:23:40	0x...a5	0x...a6	0x...1b
0x...8F	video0-0-6.mp4	2020-06-19 18:23:44	0x....a6	0x...a7	0x...1b
0x...8F	video0-0-7.mp4	2020-06-19 18:23:48	0x...a7	0x...a8	0x...1b
0x...8F	...	...	...	...	...
0x...8F	video0-0-(N-2).mp4	2020-06-19 18:28:16	0x...c1	0x...c2	0x...2c
0x...8F	video0-0-(N-1).mp4	2020-06-19 18:28:20	0x...c2	0x...c3	0x...2c
0x...8F	video0-0-N.mp4	2020-06-19 18:28:24	0x...c3	0x...c4	0x...2c
0x...8F	video.mpd			0x...ff	0x...2c

Fig. 28: Diagrama esquemático de la arquitectura propuesta para el segundo caso de uso (emisión)

En la figura 28 se muestra la arquitectura propuesta en la parte de la emisión. Un proceso de grabación genera segmentos de vídeo que van a ser procesados para extraer la información más relevante. Tras ello, se transformará en forma de bloques y se almacenará en la blockchain de pruebas y en una base de datos del servidor.

El primer paso para llevar a cabo este enfoque es conocer qué posibilidades existen a la hora de escoger una red de blockchain de pruebas. Dado que se está desarrollando un prototipo, no es factible realizar las pruebas en una red blockchain en producción. Estas redes de producción se conocen como *Mainnet* y las redes de pruebas se conocen como *Testnet*.

Tras haber investigado sobre las soluciones existentes para interactuar con redes de prueba, se elige Metamask [30] como herramienta a utilizar en la primera fase de pruebas. Metamask es un *plugin* que se añade al buscador (compatible con Google Chrome, en uso durante este proyecto)

y genera una cartera de criptomonedas con una dirección pública para utilizar tanto en la *Mainnet* de Ethereum como en diferentes redes privadas y de prueba.

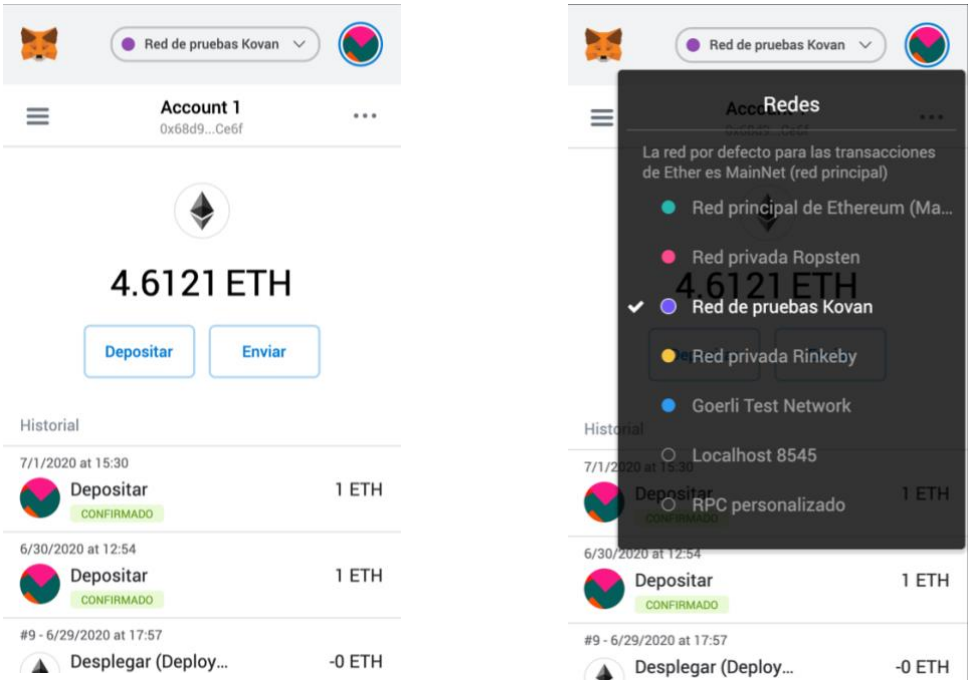


Fig. 29: Plugin Metamask en Google Chrome

Dentro de las redes de prueba que ofrece la herramienta se encuentra Ropsten, Kovan y Rinkeby. Comparando la naturaleza y comportamiento de cada una podemos elegir cuál será la mejor opción para el presente proyecto:

Ropsten: funciona bajo el algoritmo de consenso *Proof-of-Work*, esto le confiere similitud a las redes de producción de blockchain como Bitcoin o Ethereum, que también utilizan dicho algoritmo. En contraposición no es tan estable como sus competidoras por no ser inmune a ataques de spam y presenta un tiempo de minado aproximado de 30 segundos.

Kovan: funciona bajo el algoritmo de consenso *Proof-of-Authority*, lo que hace que no se asemeje tanto como Ropsten a una red en producción, pero a su favor cuenta con una serie de nodos que minan las transacciones que son nodos de confianza y presenta un tiempo de minado de entre 4 y 8 segundos aproximadamente.

Rinkeby: al igual que Kovan, utiliza *Proof-of-Authority* como algoritmo de consenso, también comparte el hecho de disponer de nodos de confianza que evitan ataques de spam y presenta un minado de 15 segundos aproximadamente.

Dado que las premisas de este enfoque se basan en conseguir registrar los segmentos tan rápido como sea posible, la red Kovan parece ser la más indicada para dicha tarea, ya que presenta el menor tiempo de minado de las tres. Por ello va a ser la red elegida en este caso de uso.

Para obtener fondos asociados a nuestra cartera existe un "grifo" que permite solicitar 1 ETH de prueba cada 24 horas. De este modo vamos a obtener los fondos necesarios para realizar las pruebas.

Una vez se dispone de una cartera con fondos, es momento de programar un Smart Contract para su posterior despliegue. El lenguaje de programación utilizado es Solidity y para este proyecto se ha hecho uso de la herramienta web Remix.ethereum [31] con el fin de programar y desplegar desde ahí los contratos a modo de prueba.

Dentro de la herramienta Remix es necesario añadir *plugins* que van a ayudar a desplegar los contratos. En este proyecto se utilizan los siguientes: *Solidity Compiler* y *Deploy & Run Transactions*.

La herramienta dispone de una interfaz de usuario muy intuitiva:

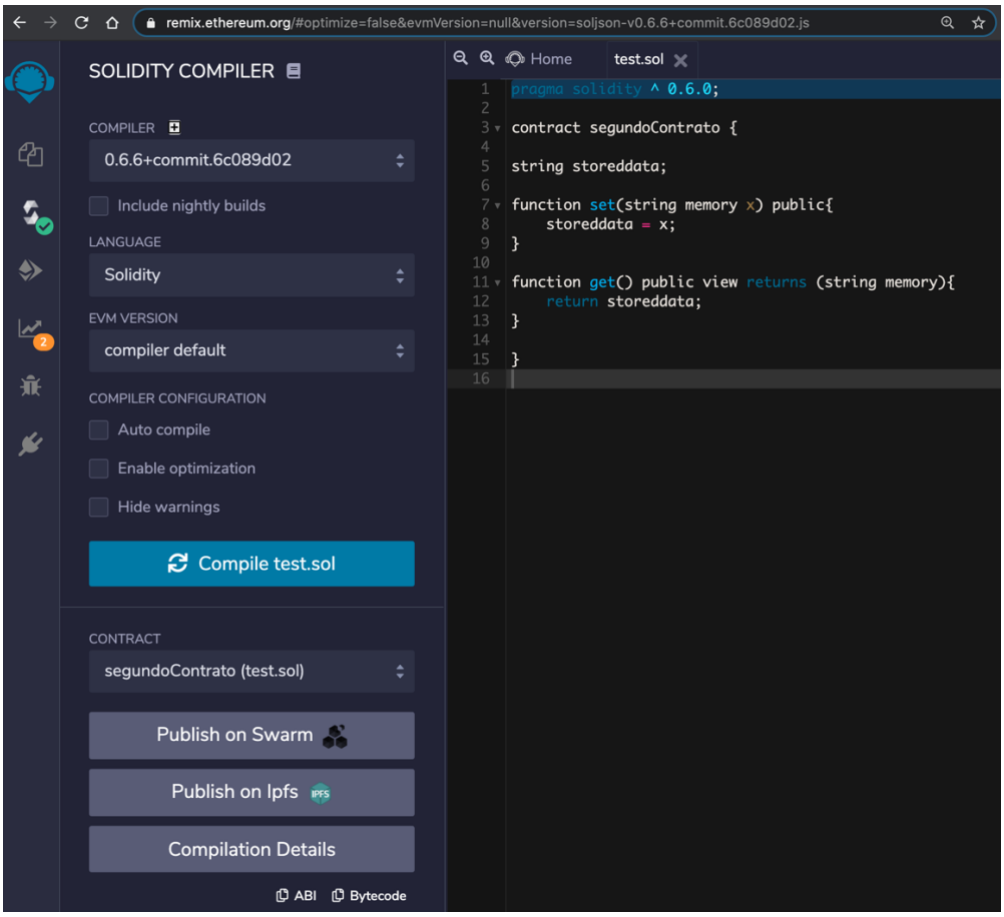


Fig. 30: Herramienta Remix.ethereum con el *plugin Solidity Compiler*

En la figura 30 se muestra el código del Smart Contract desplegado para realizar pruebas. Es en el *plugin Solidity Compiler* donde se compila el código del contrato. Esto es necesario antes de desplegarlo, ya que detecta errores de sintaxis y en ese caso, no permite desplegarlo.

El código comienza especificando la versión a utilizar de Solidity, el nombre del contrato y declarando la variable a almacenar. Este contrato de prueba es básico, ya que se compone de una función que establece el valor de una variable (líneas 7 a 9) y otra función que consulta el valor

de dicha variable (líneas 11 a 13). Para realizar pruebas es suficiente, ya que el objetivo del proyecto es simplemente almacenar valores en la blockchain y consultarlos, por ello este código suple dicha necesidad.

Una vez se ha compilado el contrato con éxito se puede desplegar. Remix y Metamask son un entorno de trabajo ideal dado que existe coordinación entre ambos. Esto quiere decir que cuando el usuario cambia de red dentro del plugin de Metamask, Remix automáticamente modifica el entorno en el que se va a desplegar el Smart Contract.

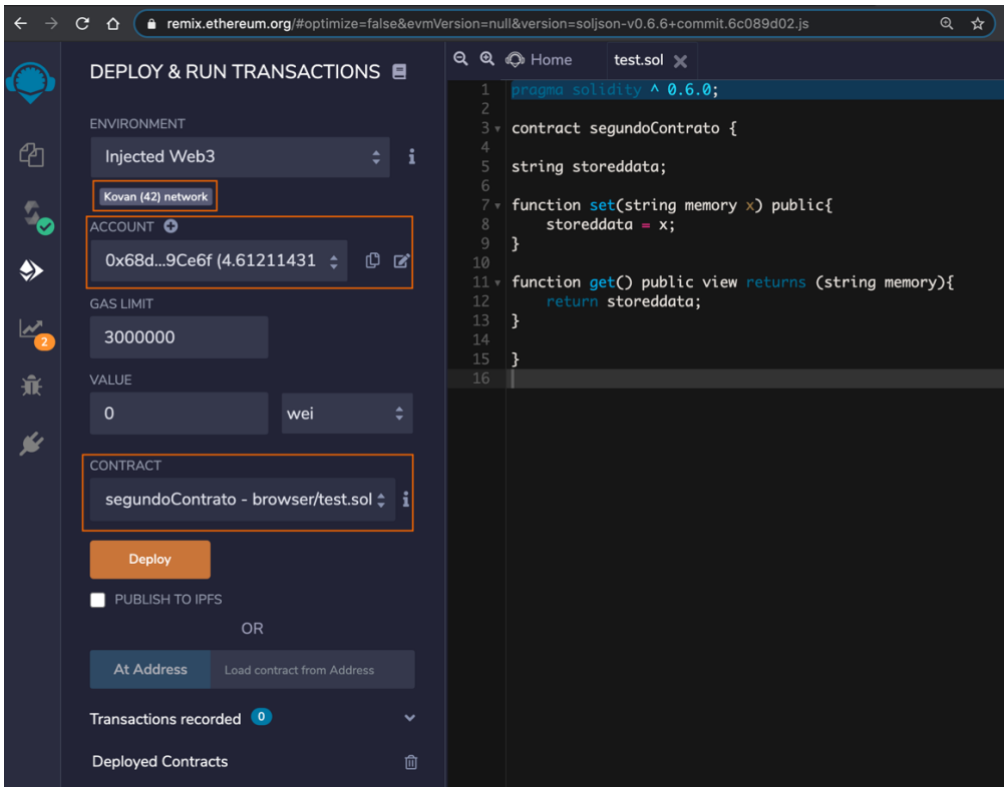


Fig. 31: Herramienta Remix.ethereum con el *plugin Deploy & Run Transactions*

Se puede apreciar en el plugin *Deploy & Run Transactions* que el entorno a desplegar es la red Kovan. Además, la dirección que lo despliega es la misma que aparece en la figura 29, lo que confirma que es la que estamos utilizando en Metamask. Por último se comprueba que el nombre del contrato coincide con el que se quiere desplegar y solo queda clicar el botón *Deploy*. Metamask muestra una ventana que solicita autorización para realizar la transacción y una vez de confirma, se registra en la red de Kovan.

El desplegar el contrato se entiende en la blockchain como una transacción con su respectivo coste de minado. De este modo se almacena de manera permanente.

Una vez se ha desplegado, se puede introducir a la blockchain de la siguiente manera:

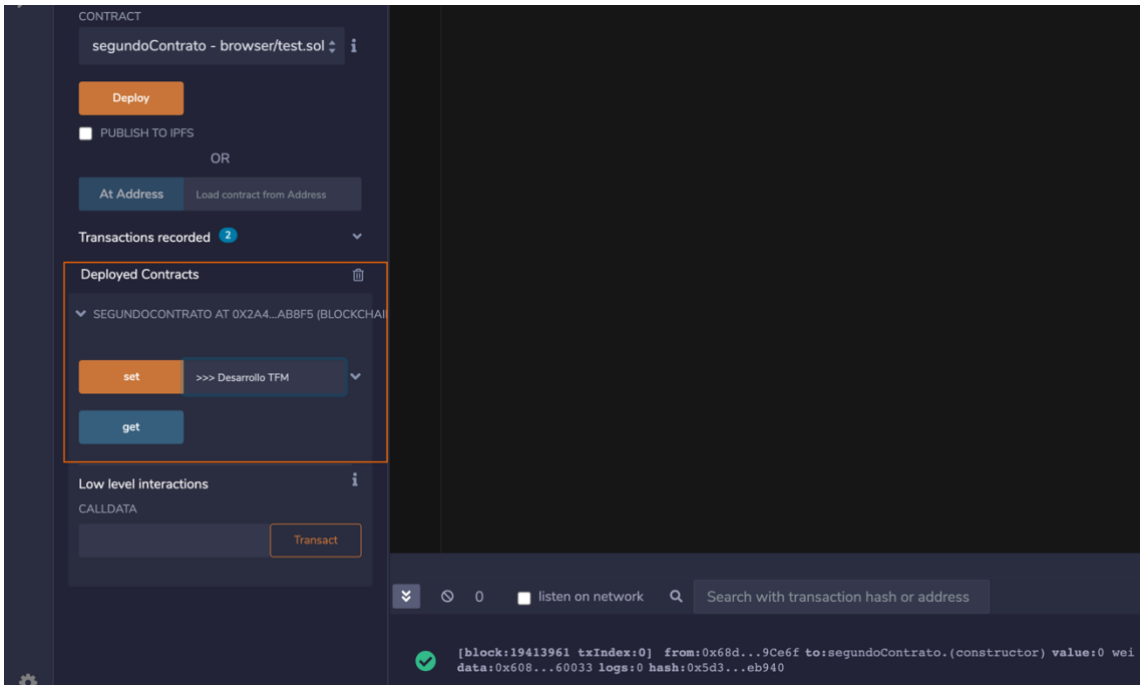


Fig. 32: Generar transacción en Remix.ethereum

Se ve como ahora aparece un apartado de contratos desplegados y los dos métodos programados, el `set` y el `get`. Si introducimos un valor dentro del `set` y clicamos, se generará una transacción. Se pueden consultar las dos transacciones hechas en https://kovan.etherscan.io/tx/ introduciendo el número de la cartera:

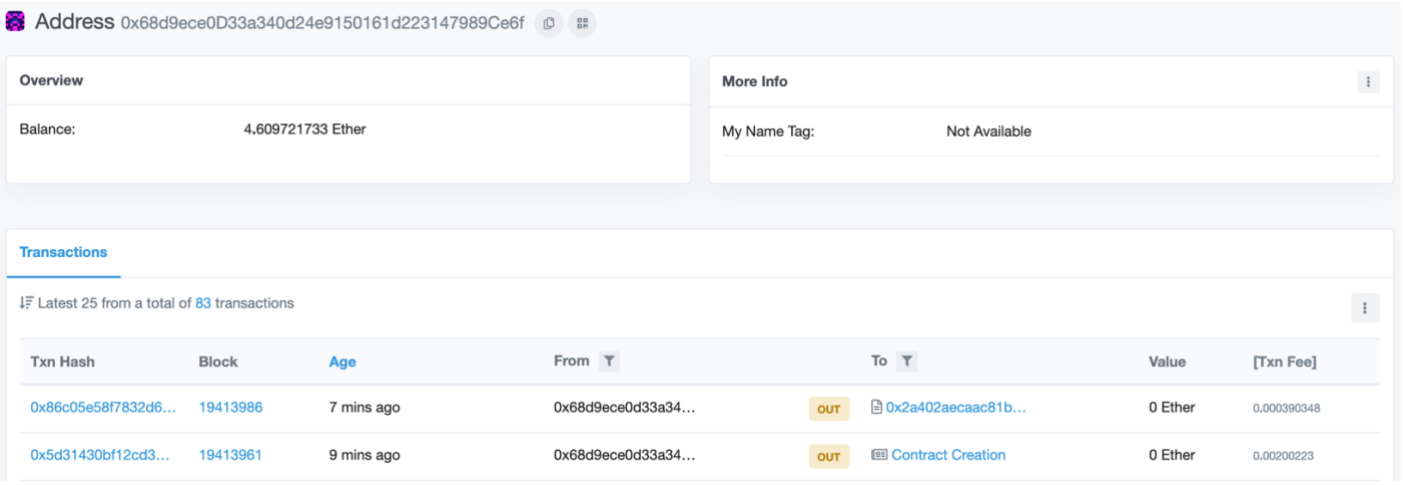


Fig. 33: Últimas transacciones de la dirección de la cartera

En la figura 33 se aprecia que existen dos transacciones, una sin dirección de destinatario, que indica la creación del contrato (la de abajo) y otra que indica la transacción que hemos realizado al clicar `set` en el Smart Contract desde Remix.

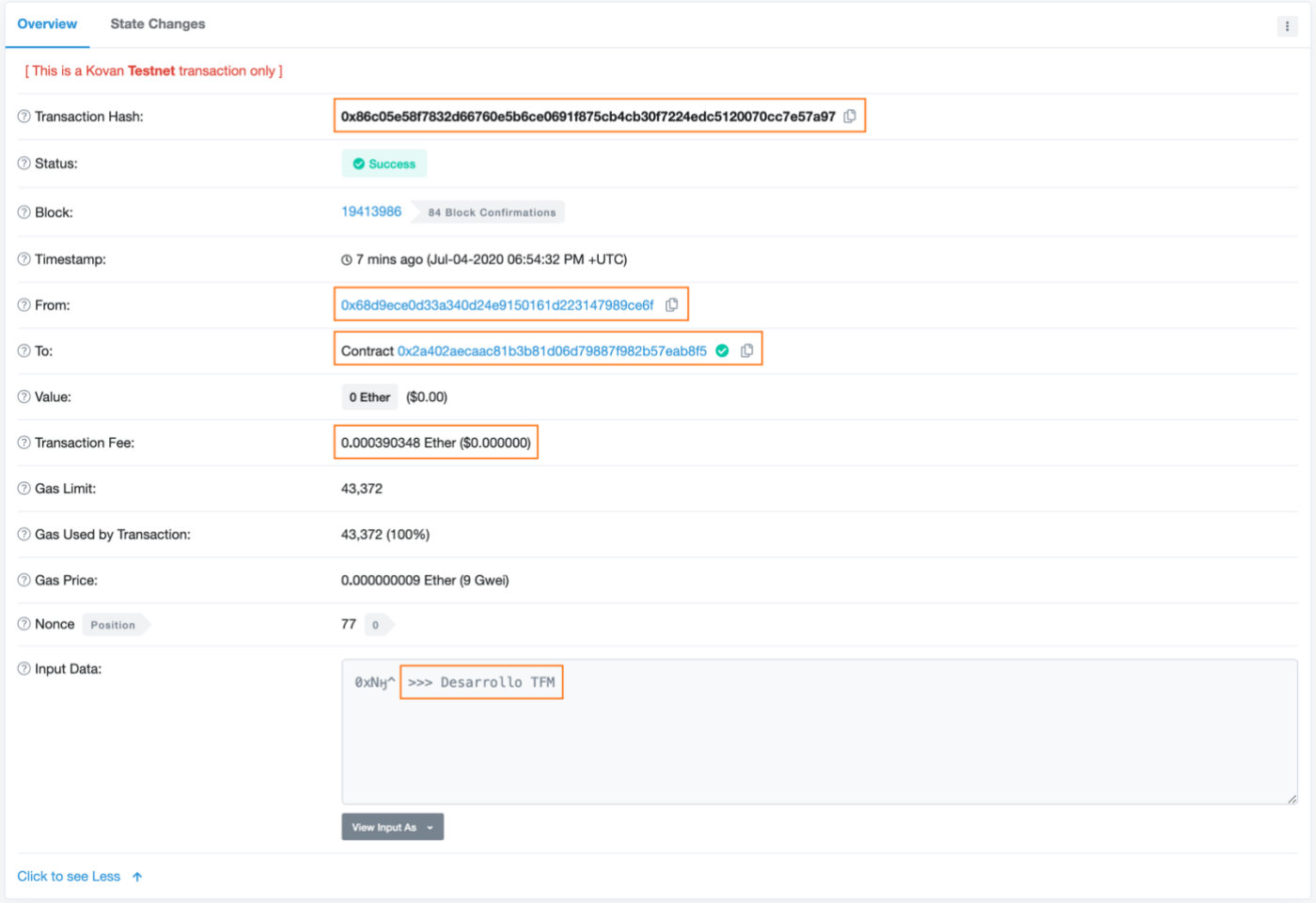


Fig. 34: Transacción realizada en la red de Kovan

Si se analiza la figura 34 de arriba a abajo, se puede apreciar el identificador de la transacción, la dirección de origen (es decir, la proporcionada por Metamask), el destino (en este caso al ser un contrato lo indica explícitamente e incluye la dirección del Smart Contract), el coste de la transacción y finalmente, lo interesante es ver que se almacenan los datos que se han introducido.

Tras una primera toma de contacto con los Smart Contracts es momento de averiguar el modo de almacenar de manera automatizada información en la blockchain.

Debido a que se pueden añadir diferentes segmentos de diferentes emisiones, se ha visto necesario modificar apartado de la grabación del vídeo, generando un identificador de la emisión único. Se ha decidido tomar como datos de entrada el nombre del usuario o empresa que realiza la emisión, el dispositivo utilizado y un timestamp, de manera que se identifica de manera única a esa emisión. Con los datos anteriores se genera un hash SHA 256 que servirá para poder identificar a qué emisión pertenecen los segmentos almacenados en la blockchain.

```
1  var ffmpeg = require('/usr/local/lib/node_modules/fluent-ffmpeg');
2  const crypto = require('crypto');
3  const fs = require('fs');
4
5  var timestamp = new Date().toISOString();
6  var id_usuario = 'salgarji';
7  var id_representacion = 'Raspberry Pi Camera Module';
8
9  var stringemision = timestamp+id_usuario+id_representacion;
10 var emision = crypto.createHash('sha256').update(stringemision).digest('hex').toString();
11 console.log(emision);
12 var buffer = new Buffer.from(emision, 'binary');
13 fs.writeFileSync('/var/www/html/segmentos/emision.txt', buffer, 'binary');
14
15 var grabacion = new ffmpeg();
16
17 grabacion.addInput('/dev/video0')
18 .inputOptions(['-y', '-nostdin', '-f v4l2', '-video_size 1280x720', '-framerate 30'])
19 .outputOptions(['-vcodec h264_omx', '-keyint_min 0', '-g 100', '-map 0:v', '-b:v 1000k', '-f dash', '-seg_duration 4',
20 '-use_template 1', '-use_timeline 0', '-init_seg_name init-video0-$RepresentationID$.mp4',
21  '-media_seg_name video0-$RepresentationID$-$Number$.mp4', '-utc_timing_url https://time.akamai.com/?iso', '-remove_at_exit 0', '-window_size 20'])
22 .output('/var/www/html/segmentos/video.mpd')
23 .run();
```

Fig. 35: Modificación del código de emisión

Se puede apreciar en las líneas 5, 6 y 7 los identificadores escogidos para este proyecto. Más adelante en las líneas 9 y 10 se calcula el hash y en la 13 se genera un archivo de texto que contiene el número de la emisión. El resto del código es idéntico al usado en el primer caso de uso.

Con este dato, el siguiente paso es automatizar el proceso de registro de información dentro de la blockchain. Para ello se va a seguir utilizando Node.js sobre JavaScript.

En el proceso de conseguir realizar una transacción de forma automatizada, es conveniente conocer el módulo `web3.js` de Node.js que permite conectarse a una API de una blockchain a través de JavaScript para realizar transacciones. Estas transacciones se envían en formato JSON-RPC, lo cual soporta el módulo, por lo que se convierte en la herramienta perfecta para realizar esta función. La instalación del módulo se recoge en el Anexo.

Para obtener una API dentro de la red Kovan existen herramientas como Infura [32] que generan una dirección para desarrollar aplicaciones de blockchain. Es la solución ideal para complementar al módulo `web3.js`.

La interfaz web muestra de manera intuitiva los enlaces generados para el usuario:

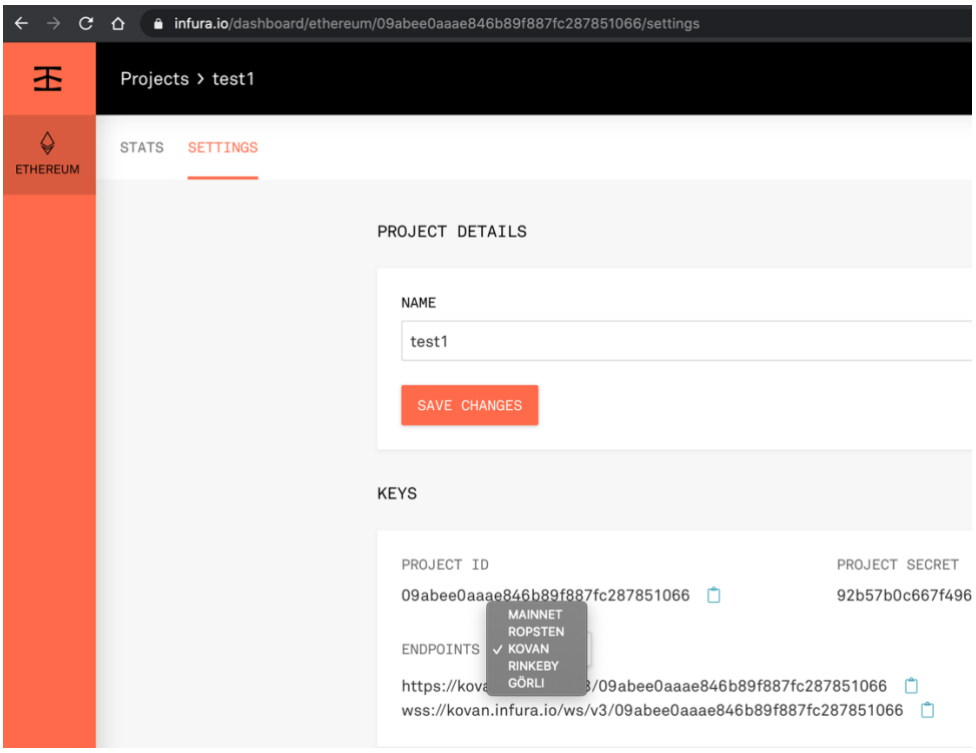


Fig. 36: Interfaz de usuario de la herramienta Infura

Además, Infura ofrece la posibilidad de trabajar con redes basadas en Ethereum tanto *Mainnets* como *Testnets*. Su versatilidad la posiciona como una de las mejores soluciones en cuanto a APIs de blockchain se refiere.

Una vez se han reunido todos los requerimientos para hacer uso de la herramienta `web3.js`, se debe investigar sobre un método sencillo de almacenar toda la información relativa a la emisión en una base de datos que pueda ser consultada a posteriori. Esta base de datos debe almacenar datos como los hashes de los segmentos y los hashes de la transacción en la cual se han almacenado en la blockchain.

Para este proyecto se ha seleccionado la base de datos más sencilla que hay disponible, SQLite, concretamente, se va a hacer uso del módulo `sqlite3` de Node.js para integrar su funcionamiento dentro del programa. Su instalación se detalla en el Anexo.

El código de la solución diseñada se ha desarrollado del siguiente modo:

```
const fs = require('fs');
const crypto = require('crypto');
const Web3 = require('/usr/src/node_modules/web3');
const sqlite3 = require('/usr/src/node_modules/sqlite3').verbose();

var prevhash ='';
var hash='';
var blockHash;
var fragNum = 1;
var Frag='';
var limit=0;
var emision;
var segHash;
var timestamp;
var cuenta = 0;
var numtrans = 0;
var hashesString='';
var segmentoArr = [];
var timestampArr = [];
var prevhashArr = [];
var hashArr = [];

var contractABI =[
    {
        "inputs": [],
        "name": "get",
        "outputs": [
            {
                "internalType": "string",
                "name": "",
                "type": "string"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "string",
                "name": "x",
                "type": "string"
            }
        ],
        "name": "set",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    }
];

const contractAddress ="0x1efec2a97D5B27d8751e8aA158F3B36A1Be0f2ba";
var almacenar ='';
const urlAPI = "https://kovan.infura.io/v3/09abee0aaae846b89f887fc287851066";
const clavePrivada  = "42AEB4C7[redacted]308D6780";
var textoAlmacenado ='';
var emisionString ='';
const precioGas     = "60000000000";
var ultimo;
var web3 = new Web3(urlAPI);
//console.log(web3.eth.accounts.privateKeyToAccount(clavePrivada).address); //muestra la clave pública
let db = new sqlite3.Database('/var/www/html/segmentos/blockchain_emision.db');
db.run('CREATE TABLE tablaBlock (emision TEXT, segmento TEXT, timestamp TEXT, prevhash TEXT, hash TEXT, transaccion TEXT)');
```

Fig. 37: Proceso de registro de hashes dentro de la blockchain (1/4)

Este programa comienza declarando las cuatro herramientas necesarias para que funcione la solución, estas son las librerías `file-system`, `crypto`, `web3` y `sqlite3`. Tras esto, se inicializan las variables que se van a utilizar y más adelante se explicará el uso de cada una de ellas. Pueden verse algunas duplicadas en comparación al método anterior.

Llama especialmente la atención la variable `contractABI` debido a su extensión. Esta variable almacena, de manera entendible para el módulo `web3.js`, el Smart Contract. La definición de ABI es *Application Binary Interface* o Inerfaz Binaria de Aplicaciones y sirve para conectar diferentes servicios de manera estándar. De hecho, en la figura 29, en la parte más baja, la propia herramienta

Remix ofrece la posibilidad de copiar el ABI de dicho Smart Contract, y ese contenido es el que se ha almacenado en la variable de la figura 35.

Se introducen variables como son el hash del Smart Contract, la dirección URL de la API provista por Infura y la clave privada de la cartera. En el caso de este proyecto, de utilizar Metamask, existe la posibilidad de conseguir la clave privada introduciendo las credenciales de nuevo en el *plugin*. Se haría del siguiente modo: Menú (tres líneas paralelas de la parte superior izquierda) > Detalles > Exportar clave privada.

En la línea 58 se introduce un precio que se paga por el gas consumido, este es un número que los mineros de la blockchain "piden" a cambio de minar el bloque. Consultando la lista de transacciones de la *Testnet* se puede estimar el valor.

En la línea 60 se crea un elemento llamado web3 a partir de la URL de la API del proyecto que va a ser utilizado en el futuro para el grueso de la transacción.

El almacenamiento, como se ha comentado se llevará a cabo gracias a SQLite y en las líneas 62 y 63 se crea una base de datos en la ruta del servidor donde se almacenan los segmentos y además se crea una tabla con seis columnas: identificador de la emisión, nombre del segmento, timestamp, hash anterior, hash actual y hash de la transacción.

El código, tras haber inicializado los valores oportunos, continúa para comenzar el procesado:

```
67  checkFlag();
68
69  function checkFlag(){
70
71      ultimo = false;
72      Frag='video0-0-'+fragNum.toString();
73      var flag = fs.existsSync('/var/www/html/segmentos/'+Frag+'.mp4');
74
75      if (flag == false){
76          if(limit < 150){
77          limit++;
78          setTimeout(checkFlag, 100);
79          }
80          else{
81              if (hashesString===''){
82              var flagmpd = fs.existsSync('/var/www/html/segmentos/video.mpd');
83                  if(flagmpd === true){
84                      ultimo = true;
85
86                      var mpdBin = fs.readFileSync('/var/www/html/segmentos/video.mpd', 'binary').toString();
87                      mpdHash = crypto.createHash('sha256').update(mpdBin).digest('hex');
88
89                      textoAlmacenado = emisionString +'Hash video.mpd: '+mpdHash+' >>> FIN DE LA EMISION';
90                      run();
91                  }
92                  else{
93                      //No se ha iniciado la grabación
94                  }
95              }
96              else{
97                  ultimo = true;
98
99                  var mpdBin = fs.readFileSync('/var/www/html/segmentos/video.mpd', 'binary').toString();
100                 mpdHash = crypto.createHash('sha256').update(mpdBin).digest('hex');
101
102                 textoAlmacenado = emisionString + hashesString +'Hash video.mpd: '+mpdHash+' >>> FIN DE LA EMISION';
103                 run();
104             }
105         }
106     }
107     else {
108         timestamp = new Date().toISOString().replace(/T/, ' ').replace(/\..+/, '').toString();
109
110         if (fragNum>1){
111             prevhash=segHash;
112         }
113         else{
114             var hashinit = fs.readFileSync('/var/www/html/segmentos/init-video0-0.mp4', 'binary').toString();
115             initHash = crypto.createHash('sha256').update(hashinit).digest('hex');
116             prevhash = initHash;
117
118         }
119         limit=0;
120         calcHash();
121     }
122 }
```

Fig. 38: Proceso de registro de hashes dentro de la blockchain (2/4)

El código sigue con un método visto en el anterior caso de uso, que es comprobar que el segmento al que se está esperando existe. Además, con el añadido de un temporizador que avisará de no existir la emisión o de haber finalizado. En este caso es más complejo al contemplar más posibilidades. En caso de existir el segmento (éxito) se salta a la línea 108 para generar un timestamp lo más rápido posible y así reflejar el tiempo de creación en el bloque. Tras esto se actualiza el valor del hash anterior, y al igual que en el anterior caso de uso, al primer segmento multimedia, se le trata de manera diferente, asignándole como hash anterior el del segmento inicial. Además, este segmento se registra en los vectores de manera aislada y directa, no automatizada.

Un cambio importante de este código con respecto al anterior es el uso de vectores para almacenar la información del nombre del segmento, timestamp, hash anterior y hash actual. Más adelante se verá más en detalle el motivo de esta decisión.

En el caso de no existir el segmento al que se espera, el código contempla tres posibilidades:

La más sencilla es que al vencer el temporizador, tener la variable `hashesString` vacía y no existir el archivo MPD, es que no se ha iniciado la grabación.

Los otros dos casos se entenderán a raíz de ver el resto del código y se explicarán más adelante.

```
127 function calcHash(){
128     var segBinario = fs.readFileSync('/var/www/html/segmentos/'+Frag+'.mp4', 'binary').toString();
129     segHash = crypto.createHash('sha256').update(segBinario).digest('hex');
130 
131     segmentoArr[fragNum]=Frag+'.mp4';
132     timestampArr[fragNum]=timestamp;
133     prevhashArr[fragNum]=prevhash;
134     hashArr[fragNum]=segHash;
135     addBlock();
136 }
137 
138 function addBlock(){
139     emision = fs.readFileSync('/var/www/html/segmentos/emision.txt', 'binary').toString();
140     emisionString = '>>> Emision: '+emision+' | ';
141 
142     if (fragNum > 1){
143         hashesString=hashesString+'Hash '+Frag+'.mp4: '+segHash+' | ';      //insertar video0-0-n
144     }
145     else{
146         hashesString='Hash init-video0-0.mp4: '+prevhash+' | Hash video0-0-1.mp4: '+segHash+' | ';
147     }
148     fragNum++;
149     cuenta++;
150 
151     if (cuenta == 4){
152         textoAlmacenado = emisionString + hashesString;
153         cuenta=0;
154         run();
155     }
156     else{
157         checkFlag();
158     }
159 }
160 
161 async function run() {
162     const web3 = new Web3(urlAPI);
163     const contract = new web3.eth.Contract(contractABI,contractAddress);
164     const account = web3.eth.accounts.privateKeyToAccount(clavePrivada);
165     const transaction = contract.methods.set(textoAlmacenado);
166     const receipt = await send(web3, account, precioGas, transaction);
167 
168     almacenar=receipt.transactionHash;
169     numtrans++;
170     guardarTodo();
171 }
172 
173 async function send(web3, account, gasPrice, transaction, value = 0) {
174     const options = {
175         to      : transaction._parent._address,
176         data    : transaction.encodeABI(),
177         gas     : await transaction.estimateGas({from: account.address, value: value}),
178         gasPrice: gasPrice,
179         nonce   : web3.eth.getTransactionCount(account.address),
180         value   : value
181     };
182     const signed  = await web3.eth.accounts.signTransaction(options, account.privateKey);
183     const receipt = await web3.eth.sendSignedTransaction(signed.rawTransaction);
184     return receipt;
185 }
```

Fig. 39: Proceso de registro de hashes dentro de la blockchain (3/4)

En este fragmento de texto empieza el procesado de datos real, es decir, los cálculos. De hecho, la función `calcHash` comienza calculando el hash del segmento actual (línea 129) y junto al timestamp generado anteriormente y el hash anterior, lo almacena en un vector para cada variable (líneas 131 a 134).

Más tarde llama a la función `addBlock`, dentro de la cual se lee el archivo de texto que contiene el identificador único de la emisión (líneas 139 y 140, estas líneas deben ir ubicadas en el mismo

lugar que se inserta la información del segmento inicial dentro de los vectores, ya que sólo necesitamos que se ejecute una vez, pero para clarificar el proceso se han colocado en este fragmento). Además, se crea una variable llamada `emisionString` que contiene el texto que se pretende introducir en la blockchain en forma de transacción.

Tras esto se comprueba si se se está procesando el primer segmento multimedia, en cuyo caso, se insertará en una variable de tipo acumulativo llamada `hashesString` además del hash de dicho segmento, el hash del segmento inicial. En caso contrario sencillamente se añadirá a la cadena ya existente el hash del segmento actual.

Tras este análisis se actualiza el valor del número de fragmento para la siguiente iteración y se suma uno a un contador llamado `cuenta`.

La existencia de este contador viene dada por la necesidad de registrar los hashes agrupados. Tal y como se ha explicado en este proyecto, la duración de los segmentos elegida a partir de los análisis de la eficiencia del sistema ha sido de 4 segundos. Recordamos que el tiempo de minado no es regular en ninguna red de blockchain y en la elegida, Kovan, el tiempo oscila entre 4 y 8 segundos pudiendo extenderse por encima de 8.

Tras diferentes pruebas se ha concluido que agrupar segmentos de cuatro en cuatro es la mejor opción, ya que agrupando en menor número se ve afectado el rendimiento. Para entenderlo mejor: si un segmento tarda 4 segundos en generarse y 8 en minarse, conforme se fuesen generando más segmentos se acumularía un retraso de manera continuada que haría que el sistema no se comportase como se busca en el objetivo de este caso de uso.

Por ello, cuando `cuenta` es igual a 4, significa que se ha almacenado tanto en los vectores como en las variables de tipo acumulativo, toda la información que se desea almacenar en la blockchain.

En ese momento, el proceso entra en la condición y se genera una variable llamada `textoAlmacenado` que combina la información sobre la emisión del vídeo y la información acumulada de los últimos 4 hashes. Esta es la información que se quiere almacenar en la red de blockchain.

Tras lo anterior se reinicia el contador a cero y se llama a la función siguiente.

Las funciones `run` y `send` funcionan de manera conjunta con el objetivo de realizar la transacción de manera firmada y automática.

La función `run` genera un objeto de tipo `web3` a partir de la URL de la API de Infura (línea 162). A partir de él es capaz de generar un objeto que contiene al Smart Contract (utilizando el ABI y la dirección introducida en una variable al inicio del programa, línea 163), genera la clave pública a partir de la clave privada que se almacena al principio del programa (línea 164), consigue formar una transacción a partir de un parámetro de entrada, que como se ha comentado, en este caso es `textoAlmacenado` (línea 165) y llama a la función send para generar un recibo de la transacción en una subrutina a la que se le pasa como parámetro los cuatro objetos generados anteriormente (línea 166).

Se ejecuta la rutina `send` (línea 173) y se le introducen los parámetros de la transacción: a quién va dirigida, los datos del contrato, una estimación del gas que se va a consumir (o sea, la parte de comisión que se descuenta de la cartera a raíz del minado), el precio del gas introducido al principio del programa a mano, el nonce, que es un índice de bloque del Smart Contract utilizado para minar el bloque y el valor de la transacción se iguala a cero en la llamada a la función debido a que no es un pago entre carteras, sino una transacción dirigida a un Smart Contract.

A partir de lo anterior y junto a la clave privada, se firma la transacción en la línea 182, para ser enviada finalmente en la línea 183 y devolver el recibo a `run`.

La función `run` continúa su ejecución desde la línea 168 donde se almacena el hash de la transacción que acaba de tener lugar y se aumenta en una unidad el contador de transacciones, que más adelante será útil a la hora de almacenar la información que se almacena. Por último se llama a la función `guardarTodo` encargada de almacenar en la base de datos todos los valores pertinentes y dejar constancia del almacenamiento en la blockchain:

```
186 function guardarTodo(){
187
188 if(ultimo == true){
189
190             segmentoArr[fragNum]='init-video0-0.mp4';
191             timestampArr[fragNum]='';
192             prevhashArr[fragNum]='';
193             hashArr[fragNum]=initHash;
194
195             segmentoArr[(fragNum+1)]='video.mpd';
196             timestampArr[(fragNum+1)]='';
197             prevhashArr[(fragNum+1)]='';
198             hashArr[(fragNum+1)]=mpdHash;
199
200     for (var i=(numtrans*4-3); i<(fragNum+2); i++){
201
202         db.serialize(function() {
203
204             var stmt = db.prepare('INSERT INTO tablaBlock VALUES (?,?,?,?,?,?)');
205
206             stmt.run(emision, segmentoArr[i], timestampArr[i], prevhashArr[i], hashArr[i], almacenar);
207
208             stmt.finalize();
209
210         });
211     }
212 }
213
214
215
216 if (ultimo == false){
217     for (var i=(numtrans*4-3); i<(numtrans*4+1); i++){
218
219         db.serialize(function() {
220
221             var stmt = db.prepare('INSERT INTO tablaBlock VALUES (?,?,?,?,?,?)');
222
223             stmt.run(emision, segmentoArr[i], timestampArr[i], prevhashArr[i], hashArr[i], almacenar);
224
225             stmt.finalize();
226
227         });
228     }
229 }
230
231     if (ultimo == false){
232         hashesString='';
233         checkFlag();
234     }
235     else{
236         db.close();
237     }
238
239 }
```

Fig. 40: Proceso de registro de hashes dentro de la blockchain (4/4)

Este proceso contempla tres casos, o lo que es lo mismo, a `guardarTodo` se puede llegar en dos situaciones diferentes. El factor determinante es si el flag `ultimo` es `true` o `false`. Este flag por defecto almacena el valor `false` ya que se la emisión está activa hasta que se detecte lo contrario. Y se cambiará a `true` cuando:

1) Se espera al siguiente fragmento, no llega y en `hashesString` no hay nada, por lo que se ha enviado el último conjunto de 4 hashes a la blockchain y no ha llegado ninguno más.

En ese caso se almacenará junto al identificador de la emisión el hash del archivo MPD (que como ya se ha comentado, se almacena el último ya que está en constante cambio)

2) Se espera al siguiente fragmento, no llega y en `hashesString` hay información. Querrá decir que no se ha llegado a cumplir la cuenta exacta de 4 hashes y se deben almacenar los últimos generados junto al identificador de la emisión y el hash del archivo MPD.

De este modo se contemplan todas las posibilidades existentes tras generar una serie de segmentos y realizar una serie de transacciones y se ordena realizar la última transacción de manera que no se pierda información.

La primera situación contemplada (línea 216) sería la de que llegue un grupo cualquiera de 4 hashes (se supone que será la situación que más repetidamente se dará) almacena del mismo modo los valores en la base de datos, pero contemplando que el inicio del bucle será el número de transacciones multiplicado por 4 y restándole 3. De este modo se almacenan los números de segmento que son múltiplos de 4 y los 3 segmentos con índice inferior sin llegar al siguiente grupo de 4. Por ejemplo, la transacción número 2 contendrá los hashes de los segmentos (2*4-3 = 5 hasta el 2*4 = 8, es decir, el 5, 6, 7 y 8, cuatro hashes) y así sucesivamente.

La segunda consiste en que, antes de finalizar la ejecución del programa, se ejecutaría el caso en que el flag `ultimo` tuviese valor `true`. En cuyo caso se almacenarían en la base de datos los segmentos comenzando en el mismo índice que la primera situación, pero terminando con el límite marcado por el número de segmento (se recuerda que este valor ha sido actualizado en la última iteración y ahora almacena un número de segmento que realmente nunca ha existido, ya que es el que se estaba esperando y no ha llegado al principio del código, por ello no hay que sumarle una unidad al valor del límite del bucle, dado que la comparación del bucle es “menor que”, ese valor es válido). Además, en este fragmento se almacena la información del segmento inicial y del archivo MPD.

Mientras el flag `ultimo` no sea true, el final de la función `guardarTodo` va a vaciar la variable `hashesString` y va a continuar esperando al siguiente segmento.

La ejecución terminará cuando el flag `ultimo` tome el valor true y se llegue al final de la función `guardarTodo`, tras cerrar la base de datos (recomendado para evitar errores).

Este método de trabajo simplifica el proceso de transferencia de los segmentos ya que no hay que procesar nada a nivel interno, sino que sencillamente, tan rápido como se generan, se almacenan.

Para comprobar que el almacenamiento se ha realizado correctamente, desde la web https://kovan.etherscan.io/tx/ podemos introducir la dirección pública provista por Metamask y buscar las transferencias hechas, ya sean de despliegue de Smart Contract o de envío de información al Smart Contract:

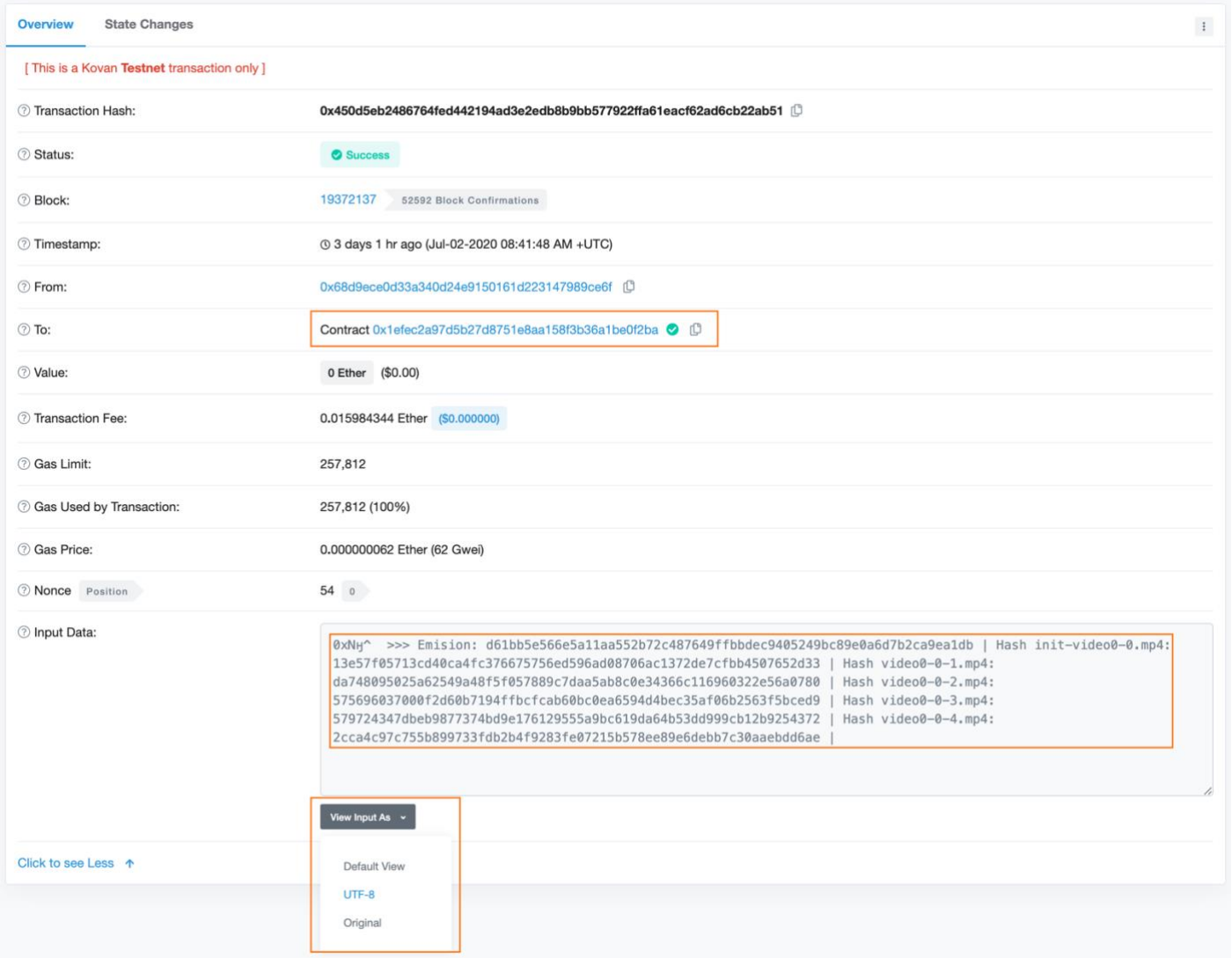


Fig. 41: Transacción reflejada en la blockchain (1/2)

En la figura 41 se puede apreciar cómo la transacción hecha desde la dirección de Metamask ha realizado una transferencia con destinatario un Smart Contract 0x1efec2a… Si nos fijamos en los datos de entrada, tras seleccionar visualizarlos como UTF-8, podemos comprobar que se trata de la primera transacción de una emisión, dado que incluye al segmento inicial y los cuatro segmentos multimedia a continuación. En caso de seleccionar los otros dos modos de visualización, se mostraría la información codificada en hexadecimal, tal y como se trata de manera interna.

Tras esta transacción se almacenarían los segmentos de cuatro en cuatro como ya se ha explicado y finalmente:

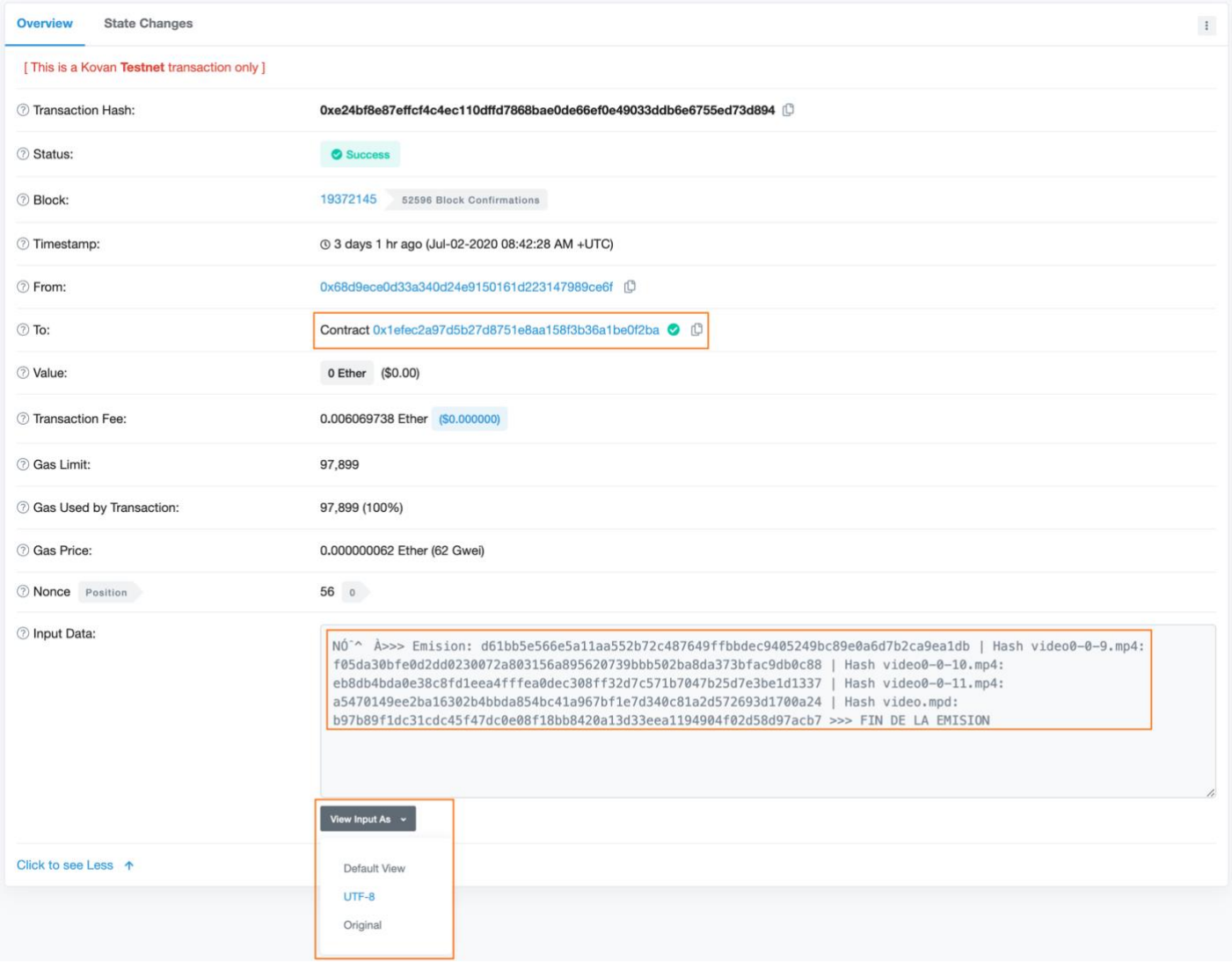


Fig. 42: Transacción reflejada en la blockchain (1/2)

En la figura 42, podemos observar cómo el Smart Contract es el mismo que en caso anterior, dado que se están realizando todas las transacciones a través de él. En este caso se ha querido mostrar cómo se formaría la última transacción blockchain en el caso de que se finalizase la emisión. De hecho, se ha dado el caso de no generarse un múltiplo de cuatro de segmentos para la última transacción y se aprecia cómo el programa soporta tal cosa enviando los últimos N fragmentos sumados al archivo MPD.

De este modo, almacenando el segmento inicial al principio y el archivo MPD al final, quedan registrados absolutamente todos los archivos que genera FFmpeg en una emisión.

6.6.2 Consumo bajo demanda y verificación de los segmentos de la emisión

Para comprobar que el registro en la blockchain se ha llevado a cabo de manera correcta se sigue un proceso de verificación análogo al 7.5.2.

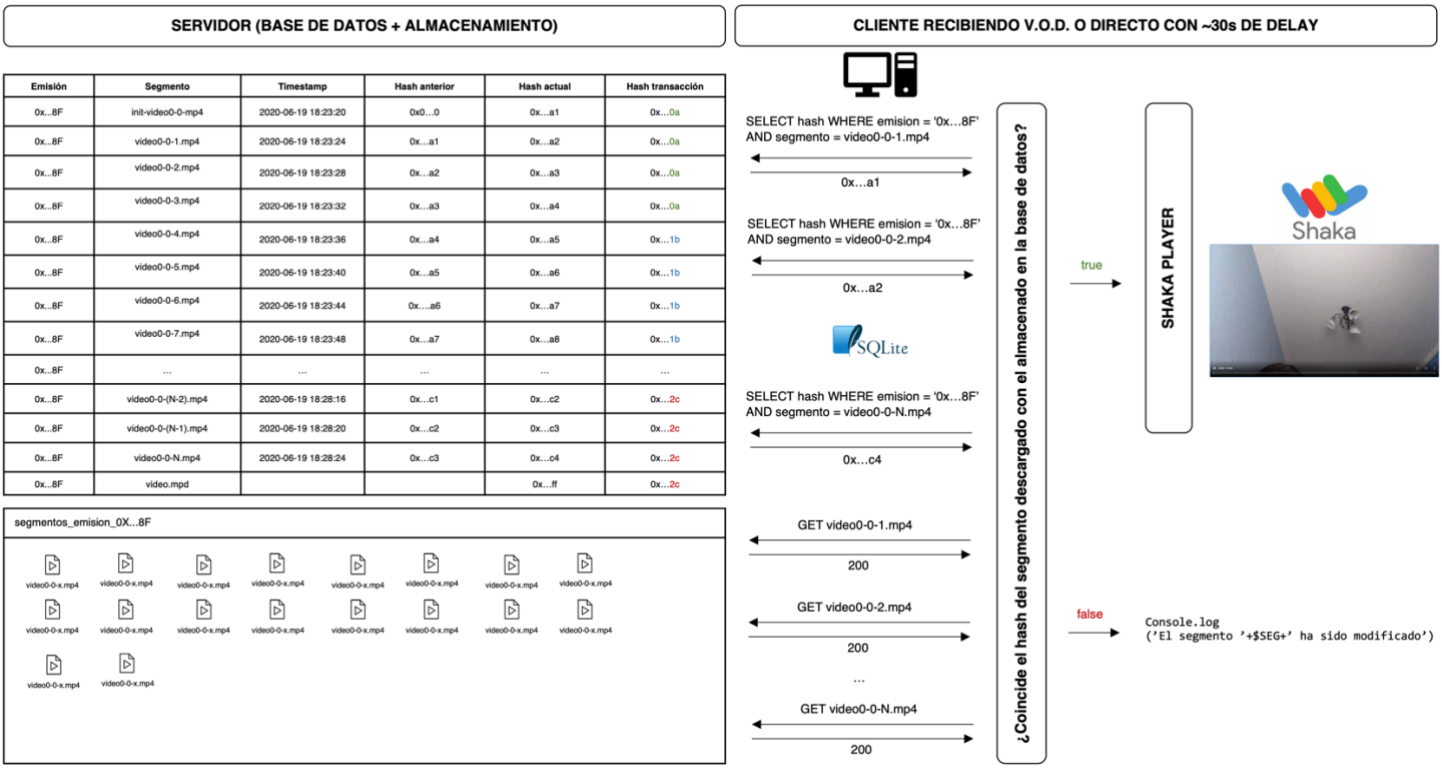


Fig. 43: Diagrama esquemático de la arquitectura propuesta para el segundo caso de uso (verificación en recepción)

En la figura 43 se muestra la arquitectura que se va a utilizar en el apartado de verificación de la información almacenada en la blockchain. Consiste en realizar una consulta a la base de datos para obtener el hash calculado de un segmento que pertenece a una emisión única. Tras la comparación entre el hash del segmento descargado y el hash almacenado en la base de datos, se determina si el segmento es válido o no. Es decir, si ha sido alterado o ha permanecido inalterado.

Este proceso se lleva a cabo utilizando un código escrito en JavaScript con herramientas de Node.js:

```
1  const http = require('http');
2  var exec = require('child_process');
3  const fs = require('fs');
4  const crypto = require('crypto');
5
6  var Frag='';
7  var i=1; //frag index
8  var index=0; //block index
9  var file;
10 var flag2;
11 var flag3;
12 var calcHash='';
13 var Frag2;
14 var limit=0;
15 var hashBBDD;
16 var emision;
17 var mpdHash;
18 var initHash;
19 var hashinitBBDD;
20 var transacinitBBDD;
21 var hashmpdBBDD;
22 var transacmpdBBDD;
23 var hashBBDD;
24 var transacBBDD;
25 var segmentoid;
26 var db;
27
28 const sqlite3 = require('/usr/src/node_modules/sqlite3').verbose();
29
30 checkUrl();
31
32 function checkUrl(){
33
34
35     Frag = 'video0-0-'+i;
36     var url = 'http://localhost:80/segmentos/'+Frag+'.mp4';
37
38     http.get(url, (res)=>{
39         var {statusCode} = res;
40         if(statusCode==200){
41             limit=0;
42             i++;
43             downloadFiles(Frag);
44             }
45         else{
46             if (limit<150){
47                 limit++;
48                 setTimeout(checkUrl, 100);
49             }
50             else{
51             }
52         }
53     }).on('error', (e)=>{
54         console.log('ERROR. File '+Frag+'.mp4 does not exist.');
55     });
56 }
```

Fig. 44: Proceso de verificación de hashes almacenados en BBDD y en la blockchain (1/4)

En las primeras líneas se llama a los módulos que se han utilizado en el proceso inverso (de almacenamiento) del punto 6.6.1. En las siguientes líneas se declaran variables e inicializan valores necesarios para el funcionamiento del programa.

En la línea 32 comienza la espera del segmento en la ruta del servidor web. En caso de recibir una respuesta 200 de tipo HTTP por el servidor, se procede a la siguiente función. En caso contrario, se espera un tiempo de *timeout*, el cual menor que en anterior caso de uso, ya que se va a realizar una verificación de vídeo bajo demanda, por lo que la tolerancia de tiempos no debería existir, aunque se inserta como margen de tiempo de que se procese correctamente la petición GET.

```
58  async function downloadFiles(){
59
60      if(i==2){
61
62              var flag0=false;
63              file = fs.createWriteStream('/var/www/html/test/video.mpd');
64              var request = http.get('http://localhost:80/segmentos/video.mpd', function(response){response.pipe(file);});
65              var endfile = new Promise(function(resolve, reject){
66                  file.on('close',()=> resolve(flag0=true));
67              })
68              let filef0 = await endfile;
69
70              var mpdBin = fs.readFileSync('/var/www/html/test/video.mpd', 'binary').toString();
71              mpdHash = crypto.createHash('sha256').update(mpdBin).digest('hex');
72
73
74              var flag1=false;
75              file = fs.createWriteStream('/var/www/html/test/init-video0-0.mp4');
76              var request = http.get('http://localhost:80/segmentos/init-video0-0.mp4', function(response){response.pipe(file);});
77              var endfile = new Promise(function(resolve, reject){
78                  file.on('close',()=> resolve(flag1=true));
79              })
80              let filef1 = await endfile;
81
82              var initBin = fs.readFileSync('/var/www/html/test/init-video0-0.mp4', 'binary').toString();
83              initHash = crypto.createHash('sha256').update(initBin).digest('hex');
84
85
86              var flagdb=false;
87              file = fs.createWriteStream('/var/www/html/test/blockchain_emision.db');
88              var request = http.get('http://localhost:80/segmentos/blockchain_emision.db', function(response){response.pipe(file);});
89              var endfile = new Promise(function(resolve, reject){
90                  file.on('close',()=> resolve(flagdb=true));
91              })
92              let filefdb = await endfile;
93
94              var flagtxt=false;
95              file = fs.createWriteStream('/var/www/html/test/emision.txt');
96              var request = http.get('http://localhost:80/segmentos/emision.txt', function(response){response.pipe(file);});
97              var endfile = new Promise(function(resolve, reject){
98                  file.on('close',()=> resolve(flagtxt=true));
99              })
100             let filetxt = await endfile;
101
102             emision = fs.readFileSync('/var/www/html/test/emision.txt', 'binary').toString();
103
104     }
105
106     flag2=false;
107
108     file = fs.createWriteStream('/var/www/html/test/'+Frag+'.mp4');
109     var request = http.get('http://localhost:80/segmentos/'+Frag+'.mp4', function(response){response.pipe(file);});
110     var endfile = new Promise(function(resolve, reject){
111         file.on('close',()=> resolve(flag2=true));
112     })
113     let filef = await endfile;
114
115     Frag2 = 'video0-0-'+(index+1).toString();
116
117     if (flag2==true){
118         //console.log('Tamaño de '+Frag+'.mp4 (bytes): '+fs.statSync('/var/www/html/test/'+Frag+'.mp4')['size']);
119         //console.log('');
120         query1();
121     }
122
123 }
```

Fig. 45: Proceso de verificación de hashes almacenados en BBDD y en la blockchain (2/4)

En este fragmento del código, cuando se procesa el primer segmento multimedia, se entra en una condición que descarga el archivo MPD, el segmento inicial, la base de datos y el archivo que contiene el número de emisión. Además, al salir de la condición se descarga el segmento multimedia actual. Esta parte del código (a partir de la línea 106) se ejecutará para cada segmento, no así desde la 60 hasta la 104, que pretende ejecutarse una sola vez, ya que es suficiente.

Una vez confirmada la descarga del segmento multimedia (línea 111), se informa del tamaño de archivo descargado por la línea de comandos y se continúa la ejecución:

```
function query1(){

        if(index==0){
        db = new sqlite3.Database('/var/www/html/test/blockchain_emision.db');
        db.get("SELECT hash,transaccion FROM tablaBlock WHERE segmento='init-video0-0.mp4'", function(err, row) {
              hashinitBBDD = row.hash;
              transacinitBBDD = row.transaccion;
              db.close();
              query2();
        });
        }else{query2();}

}

function query2(){

        if(index==0){
        db = new sqlite3.Database('/var/www/html/test/blockchain_emision.db');
        db.get("SELECT hash,transaccion FROM tablaBlock WHERE segmento='video.mpd'", function(err, row) {
              hashmpdBBDD = row.hash;
              transacmpdBBDD = row.transaccion;
              db.close();
              query3();
        });
        }else{query3();}
}

function query3(){
        db = new sqlite3.Database('/var/www/html/test/blockchain_emision.db');

        db.get("SELECT hash,transaccion FROM tablaBlock WHERE segmento='"+Frag2+".mp4'", function(err, row) {
              hashBBDD = row.hash;
              transacBBDD = row.transaccion;
              db.close();
              segName();
        });

}
```

Fig. 46: Proceso de verificación de hashes almacenados en BBDD y en la blockchain (3/4)

En esta fracción del código se realizan las consultas a la base de datos con respecto a la información almacenada. Las dos primeras funciones extraen la información del archivo MPD y el segmento inicial solo en caso de que nos encontremos en el primer índice de un contador, para que solamente se haga la consulta una vez durante la ejecución. La tercera función consulta el hash y el identificador de la transacción del segmento multimedia correspondiente en esa iteración y lo almacena en una variable global.

```
function segName(){

    if(index==0){

        console.log('Hash calculado de init-video0-0.mp4: '+initHash);

        console.log('Hash leído BBDD de init-video0-0.mp4: '+hashinitBBDD);

        if (hashinitBBDD == initHash){
        console.log('Exito. El hash descargado y el calculado para el segmento init-video0-0.mp4 coinciden.');
        }
        else{
            console.log('Error. El hash descargado no coincide con el calculado para el segmento init-video0-0.mp4.');
        }

        console.log('El archivo init-video0-0.mp4 de esta emisión ('+emision+') está registrado en la blockchain y puede consultarse
            mediante el siguiente enlace: https://kovan.etherscan.io/tx/'+transacinitBBDD+'\n');


        console.log('Hash calculado de video.mpd: '+mpdHash);

        console.log('Hash leído BBDD de video.mpd: '+hashmpdBBDD);

        if (hashmpdBBDD == mpdHash){
        console.log('Exito. El hash descargado y el calculado para el archivo video.mpd coinciden.');
        }
        else{
            console.log('Error. El hash descargado no coincide con el calculado para el archivo video.mpd.');
        }

        console.log('El archivo video.mpd de esta emisión ('+emision+') está registrado en la blockchain y puede consultarse mediante el
            siguiente enlace: https://kovan.etherscan.io/tx/'+transacmpdBBDD+'\n');

    }

    var segBinario = fs.readFileSync('/var/www/html/test/'+Frag2+'.mp4', 'binary').toString();
    calcHash = crypto.createHash('sha256').update(segBinario).digest('hex');
    console.log('Hash calculado de '+Frag2+'.mp4: '+calcHash);

    console.log('Hash leído BBDD de '+Frag2+'.mp4: '+hashBBDD);

    if (hashBBDD == calcHash){
        console.log('Exito. El hash descargado y el calculado para el segmento '+Frag2+'.mp4 coinciden.');
    }
    else{
        console.log('Error. El hash descargado no coincide con el calculado para el segmento '+Frag2+'.mp4.');
    }

    console.log('El segmento: '+Frag2+' de esta emisión ('+emision+') está registrado en la blockchain y puede consultarse mediante el
        siguiente enlace: https://kovan.etherscan.io/tx/'+transacBBDD+'\n');

    index++;

    file.end();
    file.close();

    checkUrl();

}
```

Fig. 47: Proceso de verificación de hashes almacenados en BBDD y en la blockchain (4/4)

Esta última fracción del código se encarga de comparar que los valores obtenidos en las consultas a la base de datos coinciden con los datos calculados del hash de los segmentos.

Se filtra la primera iteración para realizar la comparación del segmento inicial y del archivo MPD y más adelante en una sección de aplicación general para todas las iteraciones, se compara el hash del segmento multimedia correspondiente.

Las comparaciones pueden resultar en éxito o error, en función de si los hashes coinciden. Además, tras el mensaje de éxito o error se muestra la transacción donde quedó almacenado el hash del vídeo.

Tras esto se actualiza el número del índice utilizado para procesar el segmento y se vuelve a llamar a la primera función para esperar al siguiente segmento.

Este enfoque consigue verificar que los datos almacenados en la base de datos coinciden con los calculados por el receptor. Además, ofrece la posibilidad de consultar directamente la blockchain en caso necesario y verificar que los timestamp almacenados en la base de datos coinciden con los de las transacciones, y los hashes calculados, con los de la base de datos y a su vez con los almacenados en la blockchain. A diferencia del método anterior, en este caso no se procesa ni altera la estructura interna de los archivos.

El resultado de la ejecución sería el siguiente:

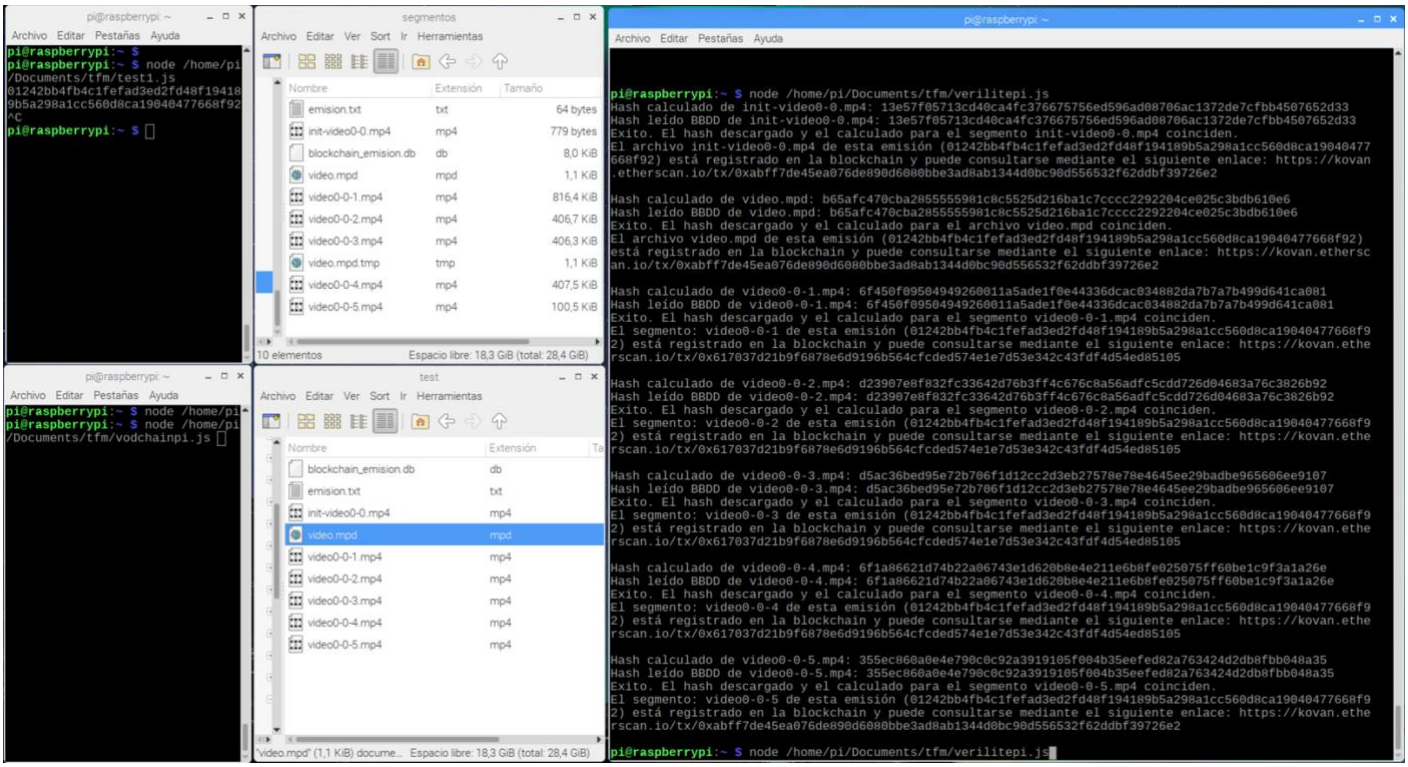


Fig. 48: Resultado del almacenamiento en BBDD y verificación de hashes de vídeo consumidos bajo demanda

Se puede apreciar cómo se realiza la comparación para cada segmento, resultando en una comparación correcta. Además, genera un enlace a la web *etherscan* donde poder consultar la transacción. Se puede ver que el funcionamiento es correcto ya que el hash inicial, el archivo MPD y el segmento número 5 comparten transacción, y los segmentos del 1 al 4 también, siguiendo el procedimiento indicado en el apartado 6.6.1.

7. Conclusiones y propuesta de trabajo futuro

A lo largo del desarrollo de este proyecto se han aplicado conocimientos de diferentes asignaturas cursadas tanto en el Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación como en el Máster Universitario en Ingeniería de Telecomunicación.

Asignaturas como Comunicación de Datos abarcan dentro de su plan de estudios la explicación y aplicación de diferentes tipos de criptografía. En relación con esto, la asignatura Seguridad ofrece una visión amplia sobre los sistemas de la información y cómo garantizar que no se produzca una vulnerabilidad. Entre los sistemas estudiados, destacan los que aseguran la integridad de la información.

El apartado multimedia está cubierto por la asignatura Sistemas Multimedia, en cuya estructura docente se encuentran contenidos sobre tratamiento de vídeo y análisis.

La programación y el diseño de servicios web se comienzan con la asignatura Programación, para continuar con Aplicaciones Telemáticas y concluye en el Máster con Integración de Servicios Telemáticos.

Con todo lo anterior, este proyecto ha conseguido lograr el objetivo que se propuso en un inicio, aportando dos soluciones al problema para diferentes casos de uso. El desarrollo de las soluciones se ha llevado a cabo tras investigar a fondo en las tecnologías y funcionamiento de las herramientas a utilizar.

Como consecuencia, se consigue adquirir una visión muy amplia de lo que supone desarrollar una herramienta o solución desde el inicio.

Este trabajo se considera un punto de partida hacia el desarrollo de diferentes proyectos, por lo que surgen las siguientes propuestas de trabajo futuro:

- Es posible generalizar el uso de la aplicación y ampliar su compatibilidad eliminando los nombres de archivos insertados a mano en el código. Mediante el análisis sintáctico del archivo MPD generado en la emisión, el cual está escrito en lenguaje XML, es posible obtener el nombre del segmento a partir de la plantilla.
- Este proyecto solamente trata archivos de vídeo, por lo que resultaría interesante añadir un micrófono al sistema y realizar la verificación del vídeo con audio.
- El proyecto, en caso de utilizarse en masa, mejoraría su eficiencia si se construyese una base de datos basada en árboles de Merkle para verificar la cadena de bloques.
- Poner a prueba el de rendimiento en diferentes *Testnets* y comparar la eficiencia en cada una, para distintos casos de uso.
- Desarrollar una interfaz de usuario que permita verificar el vídeo al mismo tiempo que se reproduce. Es decir, analizando sintácticamente, a nivel interno, el atom que contiene información temporal. Con esto y con el argumento `video.currentTime` del reproductor que muestra el instante exacto de reproducción, sería posible, al menos a priori, encontrar qué segmento se reproduce a cada momento y realizar la verificación.
- Aumentar la seguridad del primer caso de uso cifrando los atoms con una clave privada del emisor, para garantizar que sólo él ha podido cifrar la información insertada. Cualquiera podría descifrar el contenido, ya que se descifra con la clave pública.

8. Bibliografía

[1] "The Digitalization of the World from Edge to Core", David Reinsel, John Gantz, John Rydning https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf [Online]

[2] "Cisco Annual Internet Report 2018-2023", Cisco https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf [Online]

[3] "Dashcam", Wikipedia https://es.wikipedia.org/wiki/Dashcam [Online]

[4] "Tecnología china contra el coronavirus: de drones termómetro a apps que se chivan si te pones malo", El Mundo https://www.elmundo.es/tecnologia/2020/03/13/5e68a08121efa08f5b8b475c.html [Online]

[5] "Informe 0456/2015", Agencia Española de Protección de Datos https://www.aepd.es/sites/default/files/2019-12/2015-0456.pdf [Online]

[6] "*How To Time-Stamp a Digital Document*", Stuart Haber and W. Scott Stornetta https://link.springer.com/content/pdf/10.1007/BF00196791.pdf [Online]

[7] "*Bitcoin: A Peer-to-Peer Electronic Cash System*", Satoshi Nakamoto https://bitcoin.org/bitcoin.pdf [Online]

[8] "*What is Blockchain*?", Sir John Hargrave, Evan Karnoupakis

[9] "Bitcoin Price Chart", Coindesk https://www.coindesk.com/price/bitcoin [Online]

[10] "Mastering Bitcoin", Andreas Antonopoulos https://github.com/bitcoinbook/bitcoinbook [Online]

[11] Etherscan, https://etherscan.io/ [Online]

[12] "QoE matters more than QoS: Why people stop watching cat videos", Hyunwoo Nam, Kyung-Hwa Kim, Henning Schulzrinne https://ieeexplore.ieee.org/abstract/document/7524426 [Online]

[13] "Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length", Stefan Lederer (Bitmovin) https://bitmovin.com/mpeg-dash-hls-segment-length/ [Online]

[14] "Forensics Investigations of Multimedia Data: A Review of the State-of-the-Art", Rainer Poisel, Simon Tjoa

[15] "Copy-Move Forgery Detection in Images via 2D-Fourier Transform", Seniha Ketenci and Guzin Ulutas

[16] "An efficient and robust method for detecting copy-move forgery", S. Bayram, H. Taha Sencar and N. Memon

[17] Raspberry Pi 3 Model B +, https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/ [Online]

[18] Camera Module V2, https://www.raspberrypi.org/products/camera-module-v2/ [Online]

[19] FFmpeg documentation, 4.7 dash https://ffmpeg.org/ffmpeg-formats.html#dash-2 [Online]

[20] Naivechain, https://github.com/lhartikk/naivechain [Online]

[21] "HLS specification", punto 3.3 "Fragmented MPEG-4" https://tools.ietf.org/html/draft-pantos-http-live-streaming-23#section-3.3 [Online]

[22] "MP4Box.js Box Structure Viewer", Telecom Paristech https://download.tsi.telecom-paristech.fr/gpac/mp4box.js/filereader.html [Online]

[23] "Thumbcoil", http://thumb.co.il/ [Online]

[24] Bento4, https://www.bento4.com/documentation/ [Online]

[25] Hexed.it, https://hexed.it/ [Online]

[26] "Atoms, Boxes, Parents, Children & hex", Atomic Parsley http://atomicparsley.sourceforge.net/mpeg-4files.html [Online]

[27] String to Hex converter, https://codebeautify.org/string-hex-converter [Online]

[28] Chromium bug, Issue #1096273 https://bugs.chromium.org/p/chromium/issues/detail?id=1096273 [Online]

[29] Shaka Player Issues, #2631 https://github.com/google/shaka-player/issues/2631 [Online]

[30] Metamask, https://metamask.io/index.html [Online]

[31] Remix Ethereum, https://remix.ethereum.org/ [Online]

[32] Infura, https://infura.io/ [Online]

9. Anexo: Instalación de herramientas utilizadas

FFmpeg

La instalación de FFmpeg se llevaría a cabo de la siguiente manera:

```
sudo git clone https://github.com/FFmpeg/FFmpeg.git
cd ffmpeg
sudo ./configure --enable-omx --enable-omx-rpi --enable-libx264 --enable-mmal
sudo make –j4
sudo make –j4 install
```

De este modo: se clona el respositorio de FFmpeg en GitHub, se accede a la carpeta que se ha generado, se introduce la configuración que permita codificar por hardware (OpenMax), se compilan las librerías utilizando los cuatro núcleos de la CPU con -j4 y finalmente se instala la herramienta.

Fluent-FFmpeg

La instalación del módulo *fluent-ffmpeg* se lleva a cabo de la siguiente manera:

```
sudo apt-get install curl
sudo curl -sL https://deb.nodesource.com/setup_14.x | bash -
sudo apt-get install -y nodejs
cd /ruta/inmediatamente/superior/a/node_modules/
npm install fluent-ffmpeg
npm list -g --depth 0
```

Siguiendo este procedimiento se instala la herramienta Curl, muy útil para instalar software, se descargan los archivos fuente de Node.js, se instalan, y una vez se tiene Node.js instalado, se generará un directorio llamado *node_modules*. Dentro de este irán todos los módulos que se utilicen, como es el caso de FFmpeg. Se accede a la carpeta que contiene a *node_modules* y desde ahí se instala *fluent-ffmpeg*. Con el último comando se debería visualizar un listado de los módulos instalados.

NGINX

Nginx puede instalarse de manera sencilla del siguiente modo:

```
sudo apt-get update
sudo apt-get install nginx
```

Por defecto, NGINX se instala en un directorio `/var/www/html` de la Raspberry Pi.

En este momento, NGINX no está funcionando, hay que habilitar el servicio e iniciarlo:

```
sudo systemctl enable nginx
sudo systemctl start nginx
```

Bento4

Se mueve el directorio descargado al directorio deseado para su instalación y se instala del siguiente modo:

```
cd ./Bento4/Build/Targets/any-gnu-gcc
sudo make -j4
sudo make -j4 install
```

El código anterior ejecuta el archivo .mak, el cual es capaz de instalar las librerías, y así poder utilizarlas.

File System

La librería File System de Node.js, que se instala en la misma ruta que los anteriores y de manera sencilla del siguiente modo:

```
sudo npm install file-system --save
```

Crypto JS y Child Process

En el mismo directorio que los módulos instalados anteriormente, se ejecuta el siguiente código:

```
sudo npm install crypto-js --save
sudo npm install child_process --save
```

Http

El módulo se instala en la misma ruta que los demás y se instala mediante el mismo comando:

```
sudo npm install http --save
```

Web3.js

Se instala de manera sencilla del siguiente modo:

```
sudo npm install web3 --save
```

Sqlite3

Este módulo se añade del siguiente modo:

```
sudo npm install sqlite3 --save
```