Universitat de les Illes Balears

escola politècnica superior

Trabajo Fin de Grado

GRADO EN INGENIERIA TELEMÁTICA

Creación de un Smart Contract para Notificaciones Certificadas

ALEJANDRO OLIVA COLOMAR

Tutores

Maria Magdalena Payeras Capellà

Macià Mut Puigserver

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 2 de julio de 2018

ÍNDICE GENERAL

ACRÓNIMOS

TTP Trusted Third Party

TIC Tecnologías de la Información y Comunicación

P2P Peer-To-Peer

EVM Ethereum Virtual Machine

NRR No Repudio en Recepción

NRO No Repudio en Origen

SJCL Stanford Javascript Crypto Library

ABI Application Binary Interface

DoS Denial of Service

RESUMEN

En la actualidad, el correo electrónico o los servicios de mensajería instantánea son uno de los métodos de comunicación más extendidos en el mundo. Estos sistemas de comunicación pretenden, en muchos casos, imitar los servicios ofrecidos por la mensajería tradicional. Uno de estos servicios es el envío de mensajes de manera certificada. Sin embargo, en la actualidad no existe ningún estándar sobre cómo deben ser este tipo de intercambios para tener validez legal.

Este documento presenta dos propuestas de protocolos de notificaciones certificadas basados en lo expuesto en el artículo *"Notificaciones certificadas sobre blockchain"* [1], en el cual se presenta un protocolo de intercambio equitativo sin necesidad de que la Trusted Third Party (TTP) mantenga información sobre el estado en el que se encuentra el intercambio. Esto será posible mediante el uso de la tecnología *blockchain* que proporciona una estructura de datos pública, descentralizada, abierta e inmutable.

A continuación se mostrará la implementación de los dos protocolos. El primero consiste en el envío de notificaciones entre un remitente y un único destinatario, mientras que el segundo permite el envío a múltiples receptores. Finalmente, se analizarán los protocolos presentados para mostrar las propiedades aportadas y demostrar su viabilidad.

CAPÍTULO 1

INTRODUCCIÓN

Actualmente, una gran parte de las comunicaciones entre usuarios son llevadas a cabo a través de Internet. El uso de la red para estas comunicaciones supone unas ventajas claras sobre el correo tradicional, entre ellas una mayor rapidez y un menor coste económico.

No obstante, muchas de las propiedades que ofrece la mensajería tradicional no son tan fácilmente obtenibles en el formato digital. Entre ellas se encuentran las notificaciones certificadas. Una notificación certificada es aquella que permite al remitente probar que ha enviado un mensaje a uno o varios destinatarios. De la misma manera, permite demostrar que el receptor ha tenido acceso al contenido de la notificación desde una fecha determinada [1].

1.1 Notificaciones como intercambio equitativo de valores

Una notificación certificada no es más que un caso particular de intercambio equitativo de parámetros. El remitente hace entrega del mensaje al destinatario y, a cambio, obtiene un recibo que acredite la recepción. Este intercambio, pese a su simplicidad en el correo tradicional, requiere del uso de complejas operaciones criptográficas y protocolos de intercambio en el entorno digital.

Los protocolos de intercambio equitativo basan su funcionamiento en una premisa básica: no debe ser posible finalizar el intercambio hasta que todas las partes obtengan el valor deseado o, en su defecto, ninguna de ellas lo haga. Habitualmente, esta propiedad es obtenida mediante el uso de TTPs. De este modo, la tercera parte actúa como una entidad imparcial encargada de monitorizar el intercambio y garantizar que se desarrolle de manera justa [2].

En el caso de las notificaciones certificadas es importante que, al finalizar el intercambio, las dos partes cuenten con los correspondientes justificantes de no repudio en origen y destino. Estas pruebas serán obtenidas a partir del intercambio de distintos parámetros firmados digitalmente. De esta manera, el remitente no podrá negar haber

enviado el mensaje y el destinatario haberlo recibido.

1.2 Notificaciones basadas en blockchain

Las tecnologías blockchain nacen en el año 2008 de la mano de *Bitcoin*. Estas tecnologías proporcionan sistemas de almacenamiento de datos inmutables y totalmente distribuidos. Inicialmente este sistema fue creado para permitir el intercambio de monedas electrónicas sin necesidad de instituciones financieras. Haciendo uso de esta tecnología, en el año 2015 nace *Ethereum*. Esta nueva plataforma, consciente de las posibilidades que presenta la tecnología blockchain, introduce en su red Peer-To-Peer (P2P) la posibilidad de ejecutar *Smart Contracts*. Un Smart Contract es un contrato que puede ejecutarse de manera autónoma sobre una red descentralizada y que contiene las condiciones establecidas entre las distintas partes involucradas en forma de líneas de código [3].

En la sección anterior se ha expuesto como la mayoría de protocolos de intercambios equitativo basan su funcionamiento en el uso de una TTP que se encargue de gestionar el correcto transcurso de los acontecimientos. Sin embargo, el uso de una tercera parte conlleva una serie de inconvenientes, puesto que es difícil que sean confiables para todos los usuarios de la red y además pueden afectar a la eficiencia del protocolo. Por tanto, el uso de terceras partes supone un aumento del coste y del retraso en los protocolos de intercambio. Por estas razones, la tecnología blockchain, junto con los Smart Contracts, se presentan como una herramienta capaz de complementar o incluso sustituir las funcionalidades de estas entidades [4].

1.3 Objetivos y resultados esperados del proyecto

A continuación se presentarán los objetivos y resultados que se esperan obtener a partir de la realización del proyecto.

- Conocer y analizar el funcionamiento de la plataforma de código descentralizado *Ethereum* que permite el uso de Smart Contracts y aplicaciones distribuidas (*DApps*).
- Analizar los sistemas de intercambio equitativo existentes hasta el momento, especialmente los basados en Correo Electrónico Certificado o Notificaciones Certificadas.
- Diseñar y desarrollar Smart Contracts para intercambio de notificaciones certificadas entre un remitente y un único destinatario, así como otro que permita el intercambio multiparte.
- Integrar los contratos desarrollados en una aplicación web funcional que permita a los usuarios interactuar con los contratos.
- Realizar una comparativa entre las propiedades ofrecidas por los protocolos basados en blockchain respecto de los protocolos y soluciones más clásicas (sin blockchain).

- Analizar las propiedades temporales y costes de ejecución de los Smart Contract sobre la red P2P.

1.4 Estructura de la memoria

Una vez expuestos los objetivos que se pretenden alcanzar mediante la realización del presente proyecto, se detallará la estructura que seguirá la memoria. Para ello se han considerado cuatro etapas distintas:

La primera etapa, compuesta por dos secciones, consistirá en el estudio del marco de desarrollo del proyecto. En la primera sección se analizarán las propiedades y características que habitualmente se han asociado a los intercambios equitativos. Por su parte, la segunda sección explica de manera breve y sencilla el funcionamiento de la *blockchain* así como de la plataforma *Ethereum* y sus aspectos más relevantes.

La segunda etapa consistirá en el diseño de los protocolos. Esta etapa también se encontrará dividida en dos secciones. Cada una de las secciones se corresponderá con uno de los protocolos implementados. El primer protocolo mostrado permitirá el intercambio de notificaciones certificadas entre un remitente y un único destinatario. El segundo, mediante un enfoque multiparte, permitirá el envío de notificaciones certificadas a múltiples receptores.

Posteriormente, habiendo detallado ya el diseño de los dos protocolos, se procederá a mostrar la implementación de los correspondientes *Smart Contracts*. En el caso del intercambio entre un remitente y un destinatario se mostrarán dos versiones distintas de contratos, analizando la viabilidad de cada uno de ellos. Por su parte, en el caso del intercambio multiparte se mostrará un único contrato, el cual será integrado en una aplicación web funcional que cumpla con las propiedades especificadas en el diseño.

Finalmente, una vez implementado el protocolo basado en blockchain se realizará un análisis de las propiedades ofrecidas, así como de sus limitaciones. Del mismo modo, se establecerá una comparativa con los protocolos clásicos para poder ver así las mejoras que ofrece. Para finalizar, se realizará un análisis de rendimiento del protocolo, analizando los tiempos de espera y costes de ejecución de los contratos implementados y estableciendo comparaciones entre los distintos ejemplares.

CAPÍTULO 2

ESTADO DEL ARTE

En los últimos años, el auge del comercio electrónico, debido en gran parte a la generalización de las Tecnologías de la Información y Comunicación (TIC), ha revolucionado el mundo de las transacciones comerciales. Estas transacciones requieren del intercambio de valores por parte de los distintos usuarios implicados. Es por ello que desde el ámbito académico, con el objetivo de solventar la problemática generada por dichos intercambios, se han propuesto soluciones que, teniendo en cuenta las características de cada caso, tratan de solventar de manera equitativa estas interacciones.

2.1 Estudio de los Intercambios Equitativos

Los intercambios equitativos pueden ser necesarios en distintas aplicaciones electrónicas, entre las que podemos encontrar firmas electrónicas de contratos, correo electrónico certificado o pagos por recibo.

Todas estas aplicaciones tienen en común una característica principal; el intercambio debe realizarse de forma atómica. Esto significa que no debe ser posible finalizar el procedimiento mientras solo alguna de las partes haya recibido el valor esperado. De este modo, mediante un trato imparcial a todos los usuarios, se garantiza que al final de la transacción o todas las partes contarán con el elemento deseado, o en su defecto, ninguna de ellas lo hará [2]. Las soluciones propuestas para este tipo de intercambios pueden dividirse en dos principales grupos en función de si hacen uso o no de una TTP.

2.1.1 Protocolos de Intercambio Equitativo sin Terceras Partes

Los intercambios equitativos sin TTP no requieren del uso de una tercera parte de confianza que se encargue de gestionar el intercambio. En *"Protocols de comerç electrònic: Pagament anònim i intercanvi equitatiu"* [2] M. Payeras expone que este tipo de protocolos no requieren del uso de una tercera parte de confianza ya que son capaces de garantizar la seguridad del intercambio por ellos mismos. Son conocidos como

protocolos *autocontenidos*. No obstante, presentan una serie de inconvenientes que afectan directamente a su eficiencia. Esto se debe a que requieren un gran número de interacciones entre los usuarios y una complejidad de cálculo elevada.

Debido a estos factores, los protocolos *autocontenidos* no pueden garantizar la equidad en todos los escenarios. Existen protocolos basados en el intercambio gradual de información hasta conseguir el mensaje completo. Estos pueden dar lugar a intercambios no equitativos en caso de una potencia de cálculo asimétrica (un usuario puede reconstruir el mensaje con menor información que la otra parte). Por el contrario, los intercambios probabilísticos únicamente pueden garantizar la equidad dada una probabilidad específica (generalmente bastante alta).

2.1.2 Protocolos de Intercambio Equitativo con Terceras Partes

En [2] también se detallan los intercambios con TTP. En este documento se establece una clasificación que permite diferenciar entre los protocolos que requieren la intervención de la tercera parte en cada una de las fases de ejecución (*arbitrados*), de aquellos cuya intervención se produce únicamente en caso de no finalizar exitosamente (*optimistas*). En los protocolos *arbitrados*, la TTP participa en todas las ejecuciones con la intención de garantizar la seguridad en el intercambio. No obstante, esta presencia constante de la tercera parte actúa como cuello de botella en la comunicación entre usuarios. Por otra parte, los intercambios *optimistas* presentan un subprotocolo de intercambio que, en caso de realizarse de manera satisfactoria, no implica la intervención de la TTP. En caso contrario, los usuarios deberán contactar mediante otro subprotocolo con la tercera parte y aportar las pruebas pertinentes. En función de las pruebas aportadas, la TTP deberá actuar de tal manera que se garantice la equidad. A pesar de las limitaciones que provoca necesitar la intervención de una TTP, es plausible afirmar que los protocolos *optimistas* y *arbitrados* son más realistas que los *autocontenidos*.

2.1.3 Propiedades de los protocolos

Tal y como se ha establecido en las secciones previas, es deseable la inclusión de una TTP en los intercambios equitativos por motivos de eficiencia. A continuación se detallan las propiedades deseables en dichos protocolos que los autores exponen recurrentemente [1, 2, 4]:

- **Eficacia.** Si los usuarios siguen los pasos del intercambio, este se podrá llevar a cabo.
- **Equidad.** Los distintos autores distinguen entre al menos dos tipos distintos de equidad en los intercambios:
  - **Fuerte.** Cuando la ejecución de un protocolo finaliza, o bien todas las partes tienes los bienes deseados, o bien ninguna parte dispone de ellos.
  - **Débil.** Al final de la ejecución, ambas partes (o ninguna) obtienen los bienes deseados o, en su defecto, si solamente una de ellas lo ha obtenido, la parte contraria cuenta con evidencias que demuestren esta situación.
- **Asincronía.** En cualquier momento durante la ejecución del protocolo, cada parte puede, unilaterlamente, finalizar el protocolo sin que se pierda la equidad.

- **No repudio.** Después de la finalización del intercambio, cada participante debe poder probar que este se ha realizado.
  - **No repudio en origen.** Todos los participantes pueden probar el origen del mensaje que han recibido.
  - **No repudio en recepción.** Todos los participantes pueden probar que las otras partes han recibido el objeto enviado.
- **Verificabilidad de la TTP.** En caso de intervención incorrecta de la TTP, el usuario afectado debe poder demostrar el mal comportamiento de la TTP.
- **Eficiencia.** Deben evitarse costes computacionales o de comunicación excesivos y utilizarse el menor número de interacciones entre usuarios posibles.
- **Privacidad.** El contenido del intercambio debe poder ocultarse, incluso a la TTP en caso de intervención.
- **Transferibilidad de evidencias.** Las pruebas generadas por el protocolo pueden ser transferidas a entidades externas para probar el resultado del intercambio.
- **No almacenamiento de estado.** La tercera parte debe poder actuar en el intercambio sin necesidad de mantener información sobre el estado en el que este se encuentra.

2.2 Intercambios Equitativos entre dos partes

En "*An intensive survey of fair non-repudiation protocols*" [5] S. Kremer et al. realizan una revisión genérica de las soluciones clásicas de intercambios equitativos. Posteriormente J. L. Ferrer-Gomila et al. en *"Certified Electronic Mail: Properties Revisited"* [4] llevan a cabo también un análisis de las distintas propuestas existentes de intercambios equitativos entre dos partes. En el documento se analizan las propiedades ofrecidas por algunos de los protocolos más extendidos de correo electrónico certificado. A continuación se analizarán algunos de las soluciones clásicas aportadas por distintos autores.

En *"Probabilistic Non-Repudiation without Trusted Third Party"* [6] Oliver Markowitch et al. proponen un protocolo de intercambio probabilístico. Como ya se ha explicado anteriormente, este tipo de protocolos proporcionan equidad en el intercambio dada una probabilidad determinada. Sin embargo, como consecuencia de este hecho, consigue garantizar el *no repudio* sin necesidad de la intervención de una TTP. Adicionalmente, presenta las propiedades de asincronía y transferibilidad de pruebas. Por su parte, uno de los aspectos que no consigue garantizar, además de una equidad fuerte, es la eficiencia en el intercambio, ya que este necesita más de tres etapas para completarse.

Otro de los protocolos de correo electrónico certificado analizados es el propuesto en *"La Posta Elettronica Certificata"* [7] . Este protocolo de intercambio, al contrario de [6], proporciona la propiedad de *equidad fuerte*. Para conseguir esta propiedad se aleja del enfoque probabilístico y hace uso de una TTP que interviene en las distintas fases del intercambio, convirtiéndolo así en un protocolo *arbitrado*. En este protocolo se

sigue garantizando la transferibilidad de evidencias pero no la asincronía ni eficiencia del intercambio.

Por último, en *"An Efficient Protocol for Certified Electronic Mail"* [8] se propone un tercer protocolo con uso de TTP que proporciona *equidad débil*. A diferencia de [7] en este caso la TTP no interviene en todas las fases del intercambio, sino únicamente para la resolución de disputas, dando lugar así a un protocolo *optimista*. Asimismo, contrariamente a los dos casos expuestos con anterioridad, no presenta transferibilidad de evidencias, puesto que como se expone en [4], esta propiedad no es compatible con una equidad débil. Sin embargo, sí presenta eficiencia en el intercambio, ya que en el caso de no intervención de la TTP es posible finalizar satisfactoriamente la ejecución en tan solo tres pasos.

2.3 Intercambios Equitativos Multiparte

Existen muchas aplicaciones como el voto electrónico, las subastas o el correo electrónico certificado que requieren el intercambio de valores entre múltiples partes. Las topologías que pueden presentar estos intercambios son varias. Entre las mas comunes se encuentran: un emisor y múltiples receptores (1-N), múltiples emisores y receptores (N-N) o topologías de anillo. En *"Multi-Party Non-Repudiation: A survey"* [9], Jose A. Onieva et al. analizan las propiedades de estos intercambios entre las distintas partes.

Cuadro 2.1: Análisis de protocolos multiparte

Protocolo	**Aplicación**	**Propiedades**	**Inconvenientes**	**Topología**
Asokan'96	Intercambio Equitativo	- Optimista - Genérico - Equidad débil o fuerte	- Falta de eficiencia	N-N
KM'00	No Repudio Multiparte	- Esquema de cifrado de grupo - TTP ligera	- Arbitrado - Mismo mensaje	1-N
Asokan'98	Firma Contratos	- Optimista - 4 rondas	- Síncrono	Anillo
Ferrer'02	Correo electrónico Certificado	-Optimista -Eficiente -Asíncrono -TTP verificable	- Intervención de árbitro externo para disputas	1-N
ZOL'05	Correo electrónico Certificado	- Optimista - Asíncrono - Muy Eficiente	- Overhead en intervenciones de TTP	1-N

La tabla 2.1 es una versión reducida de la mostrada en [9]. En esta tabla aparecen de manera resumida las propiedades principales de algunos de los protocolos multiparte más relevantes.

Del mismo modo que en los intercambios entre dos partes, los autores no logran consensuar las propiedades que debe ofrecer cada uno de los protocolos, por lo que se opta por un compromiso entre propiedades y eficiencia, en función de la aplicación y la topología específicas. No obstante, si se aprecia una tendencia dominante en el

uso de TTPs en los protocolos multiparte. Es por ello que, con el objetivo de eliminar intervenciones innecesarias de la tercera parte, muchas de las propuestas optan por enfoques optimistas.

2.4 Intercambios Equitativos mediante Blockchain

En los últimos años, la irrupción de la tecnología blockchain ha abierto las puertas a nuevos protocolos de intercambio equitativo. Es por ello que algunos autores han decidido diseñar protocolos que hagan uso de esta tecnología para analizar las propiedades que ofrece.

En *"Towards Fairness of Cryptocurrency Payments"* [10] N. Asokan et al. utilizan por primera vez la tecnología blockchain y *Smart Contracts* para protocolos de pago equitativo. La primera propuesta consiste en un protocolo de pago síncrono. Este intercambio se lleva a cabo directamente sobre un Smart Contract, y en él comprador y vendedor intercambian una cantidad determinada de *criptomonedas* a cambio de una *firma ciega*. Este intercambio debe realizarse dentro de un intervalo temporal determinado, o de lo contrario, los fondos serán devueltos al comprador. De este modo, es el Smart Contract el encargado de gestionar el intercambio, haciendo posible así prescindir de la TTP. Otra de las propuestas realizadas en el documento consiste en un protocolo de pago equitativo optimista. En esta propuesta es posible que el intercambio se realice de manera completa sin utilizar la blockchain. Únicamente en el caso de producirse alguna disputa entre las partes será necesario acceder al Smart Contract. El objetivo de este protocolo no es eliminar la intervención de la TTP, sino complementar sus funcionalidades. De este modo se consigue que la TTP no necesite almacenar el estado del intercambio para poder resolver las disputas, puesto que es el Smart Contract el encargado de mantener esta información.

Posteriormente, M. Mut et al. proponen en *"Notificaciones Certificadas sobre Blockchain"* [1] dos protocolos similares a los especificados en [10] pero, en este caso, orientados al intercambio de notificaciones. La primera propuesta realizada consiste en el intercambio de notificaciones no confidenciales. Este protocolo es ejecutado en su totalidad sobre la blockchain, por lo que, al existir un registro público de las transacciones, el mensaje no puede ser confidencial. La característica principal de esta propuesta es la eliminación de la TTP mediante un protocolo síncrono, tal y como exponen N. Asokan et al. en [10]. La segunda propuesta es una remodelación del protocolo optimista establecido en [8] basada en tecnologías blockchain, en el cual los subprotocolos de *cancelación* y *finalización* son llevados a cabo directamente sobre el Smart Contract. De este modo, es el contrato quien debe almacenar el estado del intercambio, y ejecutar la cancelación o finalización en función de esta información.

CAPÍTULO 3

BLOCKCHAIN

En el año 2008, bajo el pseudónimo de Satoshi Nakamoto, es publicado un documento que da lugar al nacimiento de *Bitcoin*, así como también a la tecnología blockchain. Este documento sienta las bases para un tipo de red P2P de monedas electrónicas que permitiría pagos *online* sin necesidad de pasar por una institución financiera [11].

El modo de funcionamiento de los sistemas que hacen uso de blockchain se encuentran basados en la criptografía asimétrica. En estos sistemas cada usuario posee una pareja de claves pública y privada. Los usuarios son identificados únicamente por sus claves públicas y cada transacción se encuentra firmada por la clave privada del usuario que la genera. De este modo las transacciones son anónimas y, al mismo tiempo, verificables.

Cada transacción, por su parte, es retransmitida a los distintos nodos que forman la red P2P y posteriormente verificada. El problema principal subyace del hecho que cada nodo puede recibir las transacciones en un orden distinto al que han sido generadas, dando lugar así al problema principal: la posibilidad de gastar dos veces una misma moneda electrónica. Para poder evitar esta situación y hacer que toda la red acuerde un orden único de las transacciones nace la tecnología blockchain [12].

3.1 Fundamentos de Blockchain

Una blockchain es, esencialmente, una base de datos distribuida que contiene información sobre todas las transacciones o eventos que han sucedido en la red y que han sido compartidos entre todos los participantes. Dicha información no puede ser borrada bajo ningún concepto una vez queda almacenada en ella [12].

Esta blockchain, como su nombre indica, está formada por distintos bloques de información encadenados entre ellos. Estos bloques que contienen información sobre transacciones y eventos son creados por los distintos nodos que componen la red P2P (*mineros*). No obstante, esto genera la necesidad de decidir cuál de todos los posibles bloques generados por los distintos nodos será el que finalmente se incluya en la

cadena, puesto que de incluirse todos, muchas operaciones se verían duplicadas.

Esta problemática se ha resuelto mediante protocolos de consenso distribuidos. Todos los protocolos se basan en la característica principal de hacer que los nodos *mineros* realicen alguna prueba previa a la publicación del bloque. Los más utilizados son los mostrados a continuación:

- **Proof-of-Work.** Prueba de esfuerzo basada en el coste computacional de realizar una operación. En el caso de Bitcoin, esta prueba consiste en encontrar un valor determinado (*nonce*) el cual, concatenado a todas las transacciones del bloque y al *hash* del bloque anterior, proporcione un *digest* menor que un umbral determinado. El bloque cuyo nodo minero obtenga el resultado en primer lugar será introducido en la blockchain [12].

- **Proof-of-Stake.** Prueba basada en las criptomonedas que posee un nodo *minero* para otorgar el poder de publicación del próximo bloque. Esta participación puede tratarse simplemente de la cantidad total de monedas que posee cada nodo, o bien de una combinación de distintos parámetros, como por ejemplo, la antigüedad de las monedas que posee el *minero*. Este tipo de prueba será utilizado por *Ethereum* en el futuro debido a las mejoras que presenta frente a *Proof-of-Work* [13].

- **Proof-of-Burn.** Prueba en la que las probabilidades de elección de un bloque están basadas en la cantidad de monedas que el *minero* destruye. Similar a Proof-of-Work pero el coste computacional no llega a suceder [13].

- **Proof-of-Space.** Prueba similar a Proof-of-Work pero en lugar de coste computacional los mineros invierten espacio del disco [13].

3.2 Ethereum

Ethereum es una de las plataformas descentralizadas que hacen uso de tecnologías blockchain más importantes. Esta plataforma nace debido a las limitaciones que presentaba la tecnología creada por Bitcoin en 2008. Ethereum se presenta como una plataforma "*Turing Completo*", de tal manera que es capaz de permitir la creación y modificación de *Aplicaciones Descentralizadas (DApps)*. Además, otra de las características principales que presenta Ethereum es la publicación de un nuevo bloque cada 16 segundos, muy lejos de los 10 minutos que deben transcurrir en Bitcoin. De este modo, se aleja del objetivo de Bitcoin, basado únicamente en las transacciones de monedas electrónicas, y pretende dar lugar a la creación de un modelo de Internet descentralizado [14].

3.2.1 Smart Contracts

Los *Smart Contracts* son el principal factor diferencial entre las plataformas Ethereum y Bitcoin. Estos contratos inteligentes son programas informáticos cuya función principal es la realización de acuerdos virtuales para transacciones o intercambio de bienes entre usuarios de manera automatizada. Estos contratos inteligentes son los que permiten la creación de *Aplicaciones Descentralizadas*.

Los Smart Contracts permanecen almacenados en la blockchain, y a pesar de que los usuarios pueden interactuar con ellos de forma anónima, estos son públicos. Esto significa que cualquier variable o método almacenado en el contrato puede ser accedido por cualquier usuario. De este modo, junto al almacenamiento de cualquier transacción que se produzca en la blockchain, se garantiza la transparencia y fiabilidad de las operaciones.

Estos contratos son ejecutados por los nodos *mineros* en la llamada Ethereum Virtual Machine (EVM). Esta máquina virtual no es más que un sistema distribuido de ordenadores cuyos objetivos son la ejecución del código de manera segura y el control del estado interno de la blockchain. Es importante considerar que todo código ejecutado dentro de la EVM no tendrá acceso a la red externa, sistemas de archivo u otros procesos. Sin embargo, en muchos casos es necesario que un contrato tenga acceso a información procedente del exterior. Es por ello que existen los llamados *oráculos*. Estos *oráculos* se encuentran situados entre el *Smart contract* y la red externa y son los encargados de proporcionar los datos necesarios para su ejecución [14].

3.2.2 Gas

Anteriormente se ha mencionado que Ethereum se presenta como una plataforma *Turing Completo*. Si bien es cierto que permite la ejecución de Smart Contracts sobre la red P2P, dicha afirmación no es del todo cierta. La plataforma Ethereum es considerada *quasi Turing Completo*, puesto que la ejecución de los Smart Contracts se encuentra limitada. El parámetro que limita esta ejecución es conocido como *gas* y se encarga de limitar la cantidad total de cálculo que se necesita para una transacción. Esto supone un pequeño pago por parte de los usuarios para realizar operaciones, pero por otro lado, permite evitar ataques Denial of Service (DoS) mediante el uso de contratos que se ejecuten en bucle indefinidamente [14].

El precio del gas viene regulado en función de la cotización del *Ether*, la moneda electrónica de Ethereum y su unidad de medida es el *wei* (1 Ether corresponde a 10^{18} weis). La regulación entre gas y Ether se produce como medida de seguridad en caso de fluctuaciones económicas de la moneda. De este modo, si el precio del Ether se ve incrementado, el coste necesario para una unidad de gas disminuirá.

CAPÍTULO 4

PROTOCOLO DE NOTIFICACIONES CERTIFICADAS

El protocolo expuesto en esta sección pretende dar solución al intercambio de notificaciones certificadas entre dos usuarios. Este intercambio consiste en un protocolo equitativo que hará uso de una TTP como los vistos en la sección 2.1. Al tratarse de un protocolo *optimista* la tercera parte únicamente deberá intervenir en el caso de que alguno de los dos implicados en el intercambio requiera su intervención. Con la intención de reducir las funcionalidades de la TTP se utilizará la tecnología blockchain junto con Smart Contracts. El protocolo en el que se encuentra basado el presente diseño queda definido en el artículo "*Notificaciones Certificadas sobre Blockchain*"[1] el cual, a su vez, se encuentra basado en el protocolo sin blockchain expuesto en [8].

4.1 Requisitos del protocolo

En este apartado se pretenden detallar los distintos requisitos que se presuponen que deberá garantizar el protocolo. Al tratarse de un protocolo equitativo, es lógico suponer que deberá intentar satisfacer todas las propiedades que se han expuesto en la sección 2.1.3.

4.2 Partes implicadas

El objetivo principal de esta sección es la identificación de las distintas partes implicadas en el intercambio. Para ello se delimitará el rol de cada una de las entidades y se establecerán las acciones que pueden realizar.

4.2.1 Remitente

El remitente, también referenciado como A o Alice, es el emisor de la notificación certificada. Las funciones principales del remitente en el presente protocolo serán las

siguientes:

- **Iniciar la comunicación.** El remitente es el encargado de iniciar la comunicación pues es quién desea realizar el envío de la notificación.
- **Cancelar el intercambio.** El remitente, en caso de así desearlo, debe poder intentar realizar la cancelación del intercambio. El resultado de la cancelación variará en función del estado en el que se encuentre el protocolo.

4.2.2 Destinatario

El destinatario, también referenciado como *B* o Bob, es el receptor de la notificación certificada. Las funciones principales del destinatario en el presente protocolo serán las siguientes:

- **Confirmar recepción.** El destinatario es el encargado de indicar al remitente la voluntad de recibir la notificación certificada.
- **Resolver el intercambio.** El destinatario, en caso de así desearlo, debe poder intentar resolver el intercambio en el supuesto de no haber recibido satisfactoriamente la clave de descifrado del mensaje después de haber confirmado la recepción. El resultado de la resolución variará en función del estado en el que se encuentre el protocolo.

4.2.3 Trusted Third Party

La tercera parte de confianza es el último implicado en el protocolo. Como ya se ha mencionado anteriormente, la tercera parte debe actuar únicamente en la resolución de disputas entre remitente y destinatario. Entre sus funciones se encuentran:

- **Creación del Smart Contract.** La tercera parte de confianza debe realizar la creación del Smart Contract que se encargará de mantener el estado de los distintos intercambios de notificaciones.
- **Intervenir en resoluciones.** En el caso de que así se precise, la TTP deberá intervenir únicamente en las solicitudes de resolución precisadas por el destinatario de la notificación. Por otro lado, no deberá intervenir en las cancelaciones ya que serán realizadas directamente por el remitente a través del *Smart Contract*.

4.3 Preparación de las Partes

Para el diseño del protocolo es importante suponer que cada una de las partes cuenta con una serie de requisitos iniciales:

- **Wallet Ethereum.** Todos las partes implicadas deben contar con un monedero electrónico, con su correspondiente clave pública y privada, que permita al usuario interactuar con la blockchain. Es necesario remarcar que, para algunas de las funcionalidades que se realizarán, es necesario que las distintas partes (especialmente la TTP) cuenten con *Ether* (3.2.2).

- **Pareja de claves.** El presente protocolo basa su funcionamiento en las operaciones criptográficas propias de la criptografía asimétrica. Así que es necesario que cada parte cuente con una pareja de claves pública/privada para poder realizar las labores de firma y cifrado de mensajes. Cualquier sistema de criptografía asimétrica que permita estas funciones es válido, no obstante, la seguridad del protocolo dependerá también de la seguridad del sistema criptográfico.

- **Smart Contract.** Este punto es únicamente aplicable en el caso de las TTP. Todas las terceras partes deben contar con un contrato que les permita gestionar los diferentes intercambios que se producen entre los usuarios.

4.4 Notación y elementos

La notación del presente protocolo se encuentra basada en a la mostrada en [8]:

Cuadro 4.1: Notación y elementos

Notación	
X, Y	Concatenación de dos mensajes X e Y
H(X)	Función de Hash unidireccional y resistente a colisiones del mensaje X
$PR_i[H(X)]$	Firma digital sobre el digest del mensaje X con la clave privada o clave de firma
i → j: X	Envío de i hacia j del mensaje X
u ► e.f	Ejecución de la función f en el contrato e por parte del usuario u
Elementos	
SM	Smart Contract creado por la TTP
M	Mensaje certificado a enviar desde A a B
K	Clave simétrica elegida por A
$c = E_K(M)$	Cifrado simétrico del mensaje M con la clave K
$k_T = PU_T(K)$	Clave K firmada con la clave pública de la TTP
$h_A = PR_A[H[H(c), k_T]]$	Firma de A sobre la concatenación del hash de c y k_T. Primera parte de la prueba de No Repudio en Origen
$h_B = PR_B[H[H(c), k_T]]$	Firma de B sobre la concatenación del hash de c y k_T. Prueba de No Repudio en Destino
$k_A = PR_A["key", K]$	Firma de A sobre la clave k. Segunda parte de la prueba de No Repudio en Origen
$k'_T = PR_T["key", K]$	Firma de la TTP sobre la clave k. Segunda prueba de No Repudio en Origen alternativa
$h_{BT} = PR_B[H[H(c), k_T, h_A, h_B]]$	Evidencia de la solicitud de intervención a la TTP por parte de B

4.5 Fases del Protocolo

El protocolo de notificaciones certificadas se encuentra dividido en varios subprotocolos encargados de desarrollar las distintas funcionalidades del protocolo principal. A continuación se identifican y detallan cada uno de estos subprotocolos.

4.5.1 Intercambio sin TTP

Como ya se ha explicado a lo largo de esta sección se trata de un protocolo *optimista*, por tanto, las dos partes implicadas deben poder realizar el intercambio sin necesidad de acudir a la TTP. Este intercambio será realizado por una vía alternativa a la *blockchain*, como una aplicación web, con el objetivo de garantizar la confidencialidad del mensaje. El subprotocolo queda definido por el siguiente algoritmo:

Protocolo 1: Intercambio sin TTP
1. A → B: c, k_T, h_A
2. B → B: h_B
3. A → B: k_A

Como puede apreciarse en el algoritmo anterior, en caso de finalizarse sin disputas, al final del intercambio A posee h_B, o lo que es lo mismo, la prueba de No Repudio en Recepción (NRR). Por otro lado, B (el destinatario) posee tanto c como k_A. De este modo realizando la operación $D_k(c)$ puede obtener la notificación original M. Además del mensaje, el destinatario también tiene en su posesión h_A y k_A que conforman las dos partes de la prueba de No Repudio en Origen (NRO). Por lo tanto, en caso de completarse, el intercambio se habrá realizado satisfactoria y equitativamente.

4.5.2 Subprotocolo de Cancelación

El subprotocolo de cancelación es ejecutado por el emisor en el caso de no haber recibido la prueba de NRR. Este subprotoclo, diseñado para la resolución de disputas, difiere con el original mostrado en [8]. Esta diferencia se debe a que la participación de la TTP es, en este caso, prescindible debido a que sus funciones son sustituidas por las propiedades proporcionadas por la *blockchain*.

A continuación, se detalla el funcionamiento mediante el algoritmo. Esta explicación presupone que la TTP, anteriormente, se ha encargado de desplegar el contrato en la red y de ponerlo en disposición de remitente y destinatario. Además, las partes implicadas deben disponer de fondos suficientes para ejecutar las funciones necesarias en la *blockchain*.

Protocolo 2: Cancelación
1'. A ► SM.cancel
IF(finished=*true*)
2'. SM → A: h_B
ELSE
2'. cancelled=*true*

En este apartado se presentan varias diferencias con el algoritmo original de [8]. La primera de ellas es el hecho de que únicamente el remitente del mensaje, el cual se encuentra identificado por la dirección pública de su *wallet*, puede llevar a cabo la cancelación. Por lo tanto, no es necesario que verifique ante la TTP la autoría del mensaje. Además, en caso de haberse producido una finalización anteriormente, será el *Smart Contract* el que se encargue de proporcionar la prueba de NRR al remitente. La segunda razón se debe al hecho de que la *blockchain* mantiene el registro de todas las transacciones que ocurren, lo cual significa que no es necesario el uso de una solicitud de intervención de la TTP, puesto que cualquier parte puede verificar que esa cancelación se ha llevado a cabo.

4.5.3 Subprotocolo de Finalización

El último subprotocolo que se contempla en el diseño es el de finalización. Este será iniciado por el destinatario del mensaje en el caso de haber enviado la prueba de NRR (h_B), pero no haber recibido la clave k_A para poder descifrar el mensaje M.

Protocolo 3: Finalización

2'. B → TTP: $H(c), k_T, h_B, h_{BT}$
3'. TTP ► SM.finish(h_B)
3.1'. IF(cancelled=*true*)
3.1.1'. SM → TTP: "$cancelled$"
3.2'. ELSE
3.2.1'. SM almacena h_B
3.2.2'. finished=*true*
4.1'.**IF**(SM → TTP: "$cancelled$")
4.1.1'. TTP → B: $PR_T[H(\text{"}cancelled\text{"}), h_B]$
4.2'.**ELSE**
4.2.1'. TTP → B: k'_T

El protocolo de finalización, a diferencia de la cancelación expuesta anteriormente, no puede prescindir de la intervención de la TTP. Por lo tanto, el intercambio se inicia con la solicitud de intervención del destinatario a la TTP, en la que se incluyen todos los parámetros necesarios para determinar que se trata del correcto receptor del mensaje M.

Una vez comprobada la veracidad de los parámetros recibidos, es la TTP quien debe interactuar con el *Smart Contract* e intentar llevar a cabo la finalización. En el supuesto de una cancelación anterior, el contrato así se lo indicará a la TTP, quién a su vez, deberá comunicárselo al destinatario firmando dicho mensaje.

Por otro lado, si la finalización se lleva a cabo satisfactoriamente, el contrato deberá almacenar h_B para proporcionárselo al remitente en caso de futuras cancelaciones. Finalmente, la TTP proporcionará la clave k'_T al remitente para poder llevar a cabo el descifrado de M.

4.6 Resolución de disputas

Tal y como se indica en *"An Efficient protocol for Certified Electronic Mail "* [8], al finalizar la ejecución del protocolo (con o sin la intervención de la TTP), pueden surgir disputas entre los participantes en forma de repudio en origen o en recepción". La resolución de estas disputas será llevada a cabo por un árbitro externo de la siguiente manera:

- **Repudio en Origen.** *B* afirma haber recibido *M* de *A*, pero *A* niega el envío de *M*. En este caso, *B* deberá proporcionar al árbitro: M, c, k_T, h_A y k_A o k'_T en función de si ha intervenido o no la TTP. En primer lugar, el arbitro deberá confirmar si h_A es el resultado de la firma de A sobre $(H(c), k_T)$. En caso afirmativo concluirá que *A* ha enviado *c* a *B*. A continuación procederá a comprobar si k_A o k'_T corresponden con la firma de *A* o TTP respectivamente. De confirmarse, el arbitro podrá concluir que alguna de las dos partes ha enviado la clave *k* a *B*. Para concluir, comprobará si el descifrado de *c* corresponde con *M*. Si el resultado de todas las afirmaciones anteriores es correcto, el arbitro resolverá en favor de *B*. De lo contrario, la reclamación de *B* no será válida.

- **Repudio en Recepción.** *A* afirma haber enviado *M* a *B*, pero *B* niega la recepción de *M*. En este caso, *A* deberá proporcionar al árbitro: M, c, k_T, h_B, k y SM. En primer lugar, el árbitro, deberá determinar si h_B es la firma de *B* sobre $(H(c), k_T)$. De confirmarse, supondrá que *B* ha recibido *c* y k_T y que ha mostrado su voluntad en recibir el mensaje. En segundo lugar, comprobará si k_T corresponde al cifrado de *k* con la clave pública de la TTP. De este modo, determinará si *B* contaba con la posibilidad de acceder a la tercera parte para obtener la clave. No obstante, deberá primero acceder al *Smart Contract* para determinar que *A* no había realizado previamente la cancelación imposibilitando así que *B* obtuviera k_T mediante la solicitud de finalización. Es importante considerar también un comportamiento fraudulento de la TTP, de tal manera que, sin ser cierto, haya indicado a *B* que el mensaje se encontraba cancelado. En dicho caso, *B* deberá mostrar $PR_T[H("cancelled"), h_B]$. Finalmente, deberá comprobar que *c* corresponde al mensaje *M* cifrado con la clave k_T. En el supuesto de que todas las afirmaciones sean correctas, el árbitro resolverá en favor de *A*. De lo contrario, la reclamación no será válida.

CAPÍTULO 5

PROTOCOLO DE NOTIFICACIONES CERTIFICADAS MULTIPARTE

El protocolo de notificaciones certificadas multiparte supone una extensión del protocolo expuesto en el capítulo 4. La principal diferencia entre ambos es que, en este caso, se pretende ofrecer un protocolo que permita el intercambio entre dos o más usuarios. El presente diseño se encuentra directamente basado en el realizado por M. Payeras en *"Protocols de comerç electrònic: Pagament Anònim i Intercanvi Equitatiu"* [2].

5.1 Requisitos del protocolo

Como ya se ha explicado, este protocolo supone una ampliación del explicado en el capítulo 4. Por lo tanto, al seguir tratándose de un intercambio de notificaciones sobre un intercambio *equitativo* y *optimista*, los requisitos expuestos en la sección 2.1.3 deben también seguir vigentes en este protocolo y ser adaptados a las nuevas funcionalidades.

5.2 Partes Implicadas

La gran diferencia entre ambos protocolos se encuentra en las partes implicadas. Anteriormente, en el intercambio entre dos partes, se contaba con la existencia de un remitente y destinatario únicos. En este caso, se sigue manteniendo el remitente único, pero sin embargo, cada mensaje puede ir dirigido a varios destinatarios distintos, lo que implica la realización de un nuevo protocolo.

Las funcionalidades y preparación de las distintas partes no se han visto afectadas respecto de las explicaciones expuestas en las secciones 4.2 y 4.3.

5.3 Notación y elementos

A continuación se muestra la notación que será utilizada para la explicación del protocolo, siguiendo las mismas líneas generales que se han expuesto en la sección 4.4.

Cuadro 5.1: Notación y elementos

Notación	
X, Y	Concatenación de dos mensajes X e Y
H(X)	Función de Hash unidireccional y resistente a colisiones del mensaje X
$PR_i[H(X)]$	Firma digital sobre el digest del mensaje X con la clave privada o clave de firma
i → j: X	Envío de i hacia j del mensaje X
i ⇒ j:X	Envío de i hacia todos los j del mensaje X
u ► e.f	Ejecución de la función f en el contrato e por parte del usuario u
Elementos	
B	Conjunto de destinatarios del mensaje
B_i	Un destinatario particular
B'	Subconjunto de B que contiene los usuarios que acabarán el intercambio con A
B''	Subconjunto de B que contiene los usuarios que no acabarán el intercambio con A
$B'' - finalizado$	Miembros de B que han contactado con la TTP antes de la cancelación el intercambio
$B'' - cancelado$	Miembros de B que no han contactado con la antes la cancelación del intercambio
SM	Smart Contract creado por la TTP
M	Mensaje certificado a enviar desde A a B
$c = E_K(M)$	Cifrado simétrico del mensaje M con la clave K
$k_T = PU_T(K)$	Clave K firmada con la clave pública de la TTP
$h_A = PR_A[H[H(c), B, k_T, Id]]$	Firma de A sobre la concatenación del hash de c, B, k_T y Id. Primera parte de la prueba de No Repudio en Origen
$h_{Bi} = PR_{Bi}[H[H(c), k_T, Id]]$	Firma de B sobre la concatenación del hash de c, k_T y Id. Prueba de No Repudio en Recepción
$k_A = PR_A[K, B', Id]$	Firma de A sobre la clave K junto con B'. Segunda prueba de No Repudio en Origen para B_i en caso de pertenecer a B'
$k'_T = PR_T[K, B_i, Id]$	Firma de la TTP sobre la clave k para el usuario B_i. Segunda parte de la prueba de No Repudio en Origen alternativa
$h_{BiT} = PR_B[H[H(c), k_T, h_A, h_{Bi}, Id]]$	Evidencia de la solicitud de intervención a la TTP por parte de B_i

5.4 Fases del protocolo

A continuación se detallará el funcionamiento del protocolo multiparte. Para ello se explicarán en detalle los tres subprotocolos en los que se encuentra dividido el intercambio. De este modo se explicarán los pasos que conforman cada una de las fases, así como también los valores que deberán intercambiar las distintas partes.

5.4.1 Intercambio sin TTP

Del mismo modo que en el protocolo de notificaciones certificadas entre dos partes, debe ser posible que A finalice el intercambio con el conjunto de receptores B sin necesidad de recurrir a la TTP.

Protocolo 4: Intercambio multiparte sin TTP	
1. $A \Rightarrow B : c, k_T, B, h_A$	
2. $B_i \rightarrow A : k_{Bi}$	donde $B_i \in B$ y $i \in \{1, ..., \|B\|\}$
3. $A \Rightarrow B'$: K_A	

Al finalizar los tres pasos del intercambio sin TTP, cada uno de los destinatarios B' obtienen la clave utilizada para descifrar el mensaje k_A, así como la correspondiente prueba de NRO (h_A). Del mismo modo, el remitente del mensaje habrá obtenido la prueba de NRR (h_{Bi}) para cada uno de los destinatarios.

5.4.2 Subprotoclo de Cancelación

El subprotocolo de cancelación será invocado por el remitente del mensaje, a través del correspondiente *Smart Contract*, en el caso de no recibir el parámetro h_{Bi} para cada uno de los destinatarios del conjunto B. En este protocolo, al igual que en el anterior, se supone que, previamente TTP se ha encargado de la creación y distribución entre las partes implicadas del *Smart Contract*.

Protocolo 5: Cancelación multiparte
1'. A ► SM.cancel
2'. SM:
FOR (todos los $B_i \in B$")
2.1'. **IF**($B_i \in B" - finalizado$) **THEN** SM → A: h_{Bi}
2.2'. **IF**($B_i \notin B" - finalizado$) **THEN** Añadir B_i a $B" - cancelado$

En el momento de la cancelación, *A* deberá indicar todos aquellos usuarios que no han enviado h_{Bi}, o lo que es lo mismo, el conjunto B". El contrato, por su parte, se encargará de comprobar si alguno de los usuarios del conjunto especificado ha realizado una finalización anteriormente. En caso afirmativo, enviará al remitente la correspondiente prueba de NRR para ese usuario. De lo contrario, le incluirá en el grupo de usuarios cancelados ($B" - cancelado$). Por lo tanto, al final del protocolo de cancelación, el remitente *A* habrá concluido el intercambio con todos los destinatarios, ya sea satisfactoriamente o al recibir h_{Bi} o como resultado de la cancelación.

5.4.3 Subprotocolo de Finalización

El subprotocolo de cancelación será iniciado por los distintos destinatarios del mensaje, en el caso de haber enviado su correspondiente h_{Bi} pero no haber recibido la clave k_A. Dicha finalización se llevará a cabo a través de la TTP después de considerar la veracidad de los distintos parámetros recibidos.

Protocolo 6: Finalización multiparte
1'. $B_i \rightarrow$ TTP: $H(c), k_T, h_A, h_{Bi}, h_{BiT}$
2'. TTP ► SM.finish(h_{Bi})
2.1'. **IF**($B_i \in B" - cancelado$)
2.1.1' SM $\rightarrow$ TTP: "$cancelled$"
2.2'. **ELSE**
2.2.1' SM almacena h_{Bi}
2.2.2' SM añade B_i a $B" finalizado$
3.1'**IF**(SM $\rightarrow$ TTP: "$cancelled$")
3.1.1' TTP $\rightarrow B_i$: $PR_T[H("cancelled"), h_{Bi}]$
3.2'**ELSE**
3.2.1' TTP $\rightarrow B_i$: k'_T

5.5 Resolución de disputas

Según menciona M.Payeras en *"Protocols de Comerç Electrònic: Pagament Anònim i Intercanvi Equitatiu"*[2], al finalizar la ejecución del protocolo, pueden surgir disputas en la que las partes nieguen la implicación en el intercambio.

En el protocolo de notificaciones certificadas multiparte también se contemplan dos escenarios posibles: repudio en origen y repudio en recepción. Estos serán resueltos por un árbitro externo de manera distinta a las vistas anteriormente:

- **Repudio en origen.** Un receptor B_i afirma haber recibido el mensaje de A, pero este niega haberlo enviado. En primer lugar B_i debe proporcionar M, c, k_T, B, h_A y k_A o k'_T en función de si ha intervenido o no la TTP. El arbitro primeramente procederá a comprobar que el descifrado de c con la clave K corresponde con el mensaje M. En segundo lugar comprobará que h_A haya sido firmado por A y que B_i se encuentre incluido en el conjunto B. De ser afirmativo, asumirá que A envió c a B_i. Finalmente, si k_A es la firma de A sobre K i B' o k_T la de la TTP sobre K i B_i y además B pertenece a B', el arbitro concluirá que B recibió la clave K y por tanto, dará la razón a B en la reclamación.

- **Repudio en recepción.** El remitente A afirma que un remitente B_i ha recibido c y k_T, mientras que B_i niega haberlo hecho. En primer lugar A debe proporcionar al árbitro M, c, k_T, h_{Bi}, K y SM. El árbitro comprobará inicialmente si h_{Bi} es la firma de B_i sobre $((H(c), k_T, Id)$, de ser así asumirá que B_i recibió correctamente c y k_T, y que por tanto, contaba con la posibilidad de obtener la clave K. Adicionalmente, comprobará que k_T corresponda con la clave K cifrada con la clave pública de la TTP. En caso de demostrarse todas estas evidencias, el árbitro deberá proceder a

comprobar en el *Smart Contract* que no se hubiese producido una cancelación que impidiese a B_i realizar la finalización del intercambio. Es importante considerar también la posibilidad de que la TTP haya actuado de forma fraudulenta, y haya indicado a B que el mensaje se mostraba cancelado y no fuese esto cierto. En dicho caso, B deberá presentar $PR_T[[H("cancelled"), h_{Bi}]]$ para demostrar este comportamiento. En caso de no contar con dicho justificante, el árbitro deberá comprobar que c corresponde al mensaje M cifrado con la clave K. De ser así, concluirá que B_i si recibió el mensaje de A.

CAPÍTULO 6

IMPLEMENTACIÓN

La fase de implementación consiste en el desarrollo de una aplicación completamente funcional que permita el intercambio de notificaciones entre distintos usuarios. Inicialmente se presentará la realización del Smart Contract para el protocolo de notificaciones certificadas entre dos partes. En este apartado se considerarán las dos variantes que han surgido durante la fase de implementación y se determinará cual se ajusta mejor al diseño inicial. Seguidamente, se presentará la solución escogida para el protocolo multiparte junto con su implementación sobre una aplicación web que englobe todas las funcionalidades del protocolo.

6.1 Decisiones de Implementación

Las decisiones de implementación tomadas en este apartado han tenido como objetivo permitir el desarrollo de una aplicación funcional basada en los diseños vistos en los capítulos 4 y 5. Asimismo, se han tenido en cuenta aspectos como la funcionalidad práctica en un entorno real o la facilidad de uso para los usuarios finales.

6.1.1 Tecnologías seleccionadas

La implementación de soluciones basadas en la *blockchain* es un tema emergente, y por lo tanto, se trata de un entorno cambiante y en el que no existen unos estándares tecnológicos a utilizar. Teniendo en cuenta la falta de un estándar, y con el objetivo de garantizar la facilidad de uso y la funcionalidad independientemente de la plataforma del usuario, se ha decidido hacer uso de una arquitectura *web*.

En esta sección se separarán las tecnologías en tres grupos: *front-end*, *back-end* y *blockchain*.

- *Front-end*.
  - **HTML.** Lenguaje de marcas utilizado para determinar la estructura y contenido de una página web.

  - **JavaScript.** Lenguaje de programación orientado a objetos para desarrollo web ejecutado por un navegador.

- *Back-end:*

  - **PHP.** Lenguaje de programación para desarrollo web ejecutado en el lado del servidor y encargado de realizar las comunicaciones con la BD.
  - **MySQL.** Sistema de gestión de bases de datos relacionales. Permite el almacenamiento, modificación, eliminación y obtención de datos mediante el uso de comandos SQL.

- *Blockchain:*

  - **Solidity.** Lenguaje de programación basado en JavaScript, Python y C++ utilizado para interactuar con la EVM, y por tanto, utilizada para el desarrollo de Smart Contracts [15].
  - **Remix.** Herramienta *online* para el desarrollo y despliegue de Smart Contracts. Debido a que la compilación automática de contratos se ha considerado fuera del alcance del proyecto, Remix se ha utilizado como la herramienta de compilación de los distintos *Smart Contracts* desarrollados [16].
  - **Web3.js** API de JavaScript que permite la interacción y despliege de Smart Contracts sobre la *blockchain* de Ethereum mediante el uso de la librería JSON-RPC. Esta librería permite interactuar directamente con un nodo de la red P2P de Ethereum.
  - **Metamask** Extensión para navegadores web que funciona como enlace entre el navegador y la blockchain *Ethereum*. Permite la gestión de *Wallets*, así como también la ejecución *aplicaciones distribuidas* sin la necesidad de instalar un nodo completo de la red P2P de Ethereum en la máquina local [17].

6.1.2 Sistema de criptografía

Tal y como se ha explicado en la fase de diseño, los dos protocolos necesitan del uso de sistemas de criptografía. Para realizar la implementación del correspondiente sistema se ha hecho uso de la Stanford Javascript Crypto Library (SJCL) [18]. Esta librería criptográfica permite la implementación de cifrados mediante algoritmos de claves simétricas y asimétricas, así como también la realización de firmas digitales, todo ello mediante el uso de JavaScript.

A continuación, se explicarán los algoritmos criptográficos y sus distintos usos:

- **AES.** AES es un sistema criptográfico de cifrado por bloques mediante el uso de una clave simétrica. En este preciso caso, se ha hecho uso de una clave de 128 bits con el modo de cifrado CCM. El uso principal de AES en la implementación actual corresponde al cifrado del mensaje a intercambiar M mediante la clave K.

- **EC-ElGamal.** Algoritmo de cifrado de clave pública basado en el intercambio de claves *Diffie-Hellman* que hace uso de curvas elípticas para la generación de

claves. Para dicha generación de claves se ha hecho uso de una curva P-256, considerada computacionalmente segura hasta el año 2030 [18]. El protocolo ElGamal se ha utilizado para el cifrado y descifrado de la clave simétrica *k*, mediante las claves públicas y privadas de la TTP.

- **ECDSA** Algoritmo criptográfico de firmas digitales, variante de *DSA*, que hace uso de la criptografía de curvas elípticas. Del mismo modo que para el protocolo *EC-ElGamal*, se ha hecho uso de una curva P-256. El protocolo *ECDSA* se ha utilizado para aplicar firmas digitales a distintos parámetros del intercambio. Entre ellos encontramos las pruebas de NRR y NRO que se han expuesto en el diseño.

Es importante notar que inicialmente los protocolos de intercambio estaban diseñados para hacer uso de una única pareja de claves de criptografía asimétrica. Esta clave sería utilizada tanto para firmar mensajes como para permitir el intercambio de la clave simétrica AES. No obstante, se han utilizado dos parejas de claves. La pareja de claves del algoritmo EC-ElGamal se utilizará para el intercambio de la clave simétrica, mientras que la correspondiente al algoritmo ECDSA será utilizada par la firma de mensajes. Esta decisión se encuentra basada en la imposibilidad de encontrar una librería criptográfica Javacscript que permitiese realizar operaciones de cifrado y firma con la misma pareja de claves.

Finalmente, es imprescindible asumir que, a pesar de la seguridad que presente la librería SJCL, el uso de funciones criptográficas a través de Javascript en entornos web no ofrece una seguridad equiparable a la usada en aplicaciones de escritorio. Esto se debe a la posibilidad de ataques de inyección de código, a la existencia de servidores maliciosos, o incluso, a ataques de canal lateral. [18]

6.2 Arquitectura

Antes de empezar con la implementación es de fundamental importancia delimitar la arquitectura sobre la cual se realizará. La Figura 6.1 muestra de manera esquemática y simplificada esta arquitectura.

Tal y como puede apreciarse, se trata de una arquitectura cliente-servidor clásica, pero con una ligera variación. El cliente web se comunicará con el servidor remoto mediante el protocolo HTTP /HTTPS. Este servidor procesará las peticiones recibidas, y en caso de ser necesario, accederá a la correspondiente base de datos mediante SQL. Finalmente, enviará la respuesta al cliente. No obstante, en esta arquitectura se encuentran dos participantes más: *Metamask* y la blockchain de *Ethereum*. Esto se debe a que algunas de las funcionalidades de la aplicación, como ya se ha explicado, se realizarán mediante el uso de Smart Contracts. Es por ello que es necesario que el cliente sea capaz de conectarse por si mismo a la blockchain. Para permitir esta conexión es necesario el uso de *Metamask*, ya que, tal y como se ha explicado en la sección 6.1.1, esta extensión permite a navegadores convencionales la interacción con la red P2P de Ethereum, permitiendo así la ejecución de aplicaciones distribuidas (DApps). Es importante comprender que todas las acciones realizadas sobre el Smart Contract se llevarán a cabo de manera directa sobre un nodo de Ethereum. De este

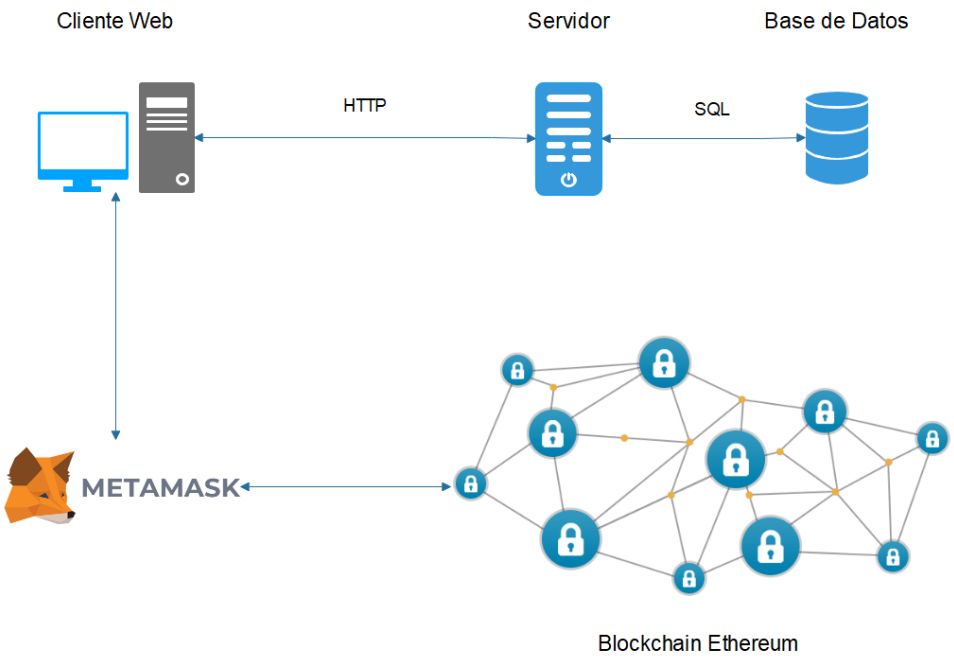

Figura 6.1: Arquitectura de la aplicación

modo, puede verse como dos tecnologías, aparentemente contrarias, como son las arquitecturas cliente-servidor y las redes P2P funcionan de manera complementaria.

6.3 Relación entre entidades

A la hora de implementar el protocolo, debe considerarse como serán las relaciones entre las distintas entidades que participan en el intercambio. En la Figura 6.2 quedan definidas estas relaciones.

- Un *Remitente* puede enviar un mensaje a uno o varios *destinatarios* (en función de si se trata del protocolo simple o multiparte).
- Un *destinatario* puede enviar la prueba de NRR a un *remitente* que ha enviado un mensaje.
- Un *remitente* puede enviar la clave de cifrado del mensaje a un *destinatario* que haya enviado la prueba de NRR.
- Un *remitente* puede cancelar el envío del mensaje mediante el acceso al *Smart Contract*.
- Un *destinatario* puede solicitar a la *TTP* la finalización del intercambio, en caso de no haber recibido la clave de cifrado.

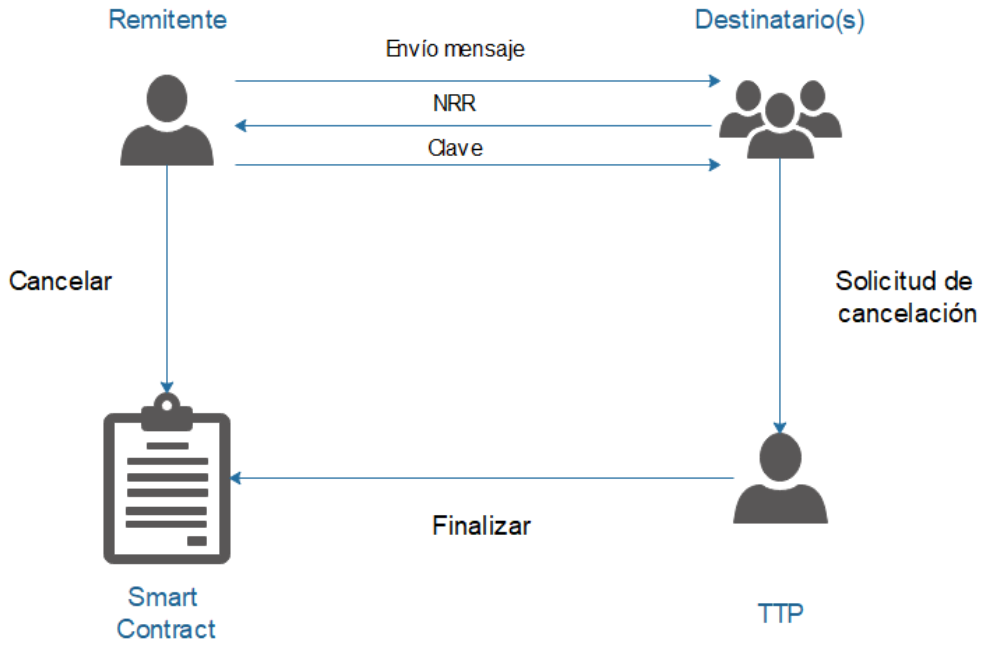

Figura 6.2: Entidades de la aplicación

- La *TTP* puede llevar a cabo la finalización de un intercambio a través del *Smart Contract* si así lo ha solicitado un *remitente*.

6.4 Diseño de la Base de Datos

El diseño de la base de datos se ha realizado teniendo en cuenta los distintos parámetros indicados en la fase de diseño de los protocolos en los capítulos 4 y 5.

6.4.1 Base de Datos para Notificaciones Certificadas

A continuación se detallará el diseño de la base de datos utilizada para el intercambio simple. Por su parte, el modelo relacional entre las distintas tablas puede verse en la figura 6.3.

- **Usuario.** Contiene la información fundamental para un remitente o destinatario: credenciales, dirección pública de su *wallet* y las claves públicas ElGamal y ECDSA.
- **TTP.** Contiene la información fundamental para una tercera parte: credenciales, dirección pública de su *wallet*, claves públicas ElGamal y ECDSA y dirección del Smart Contract que gestiona.
- **Mensaje** Contiene los parámetros del intercambio accesibles y comunes para todas las partes: c, h_c, k_T y la fecha de envío.

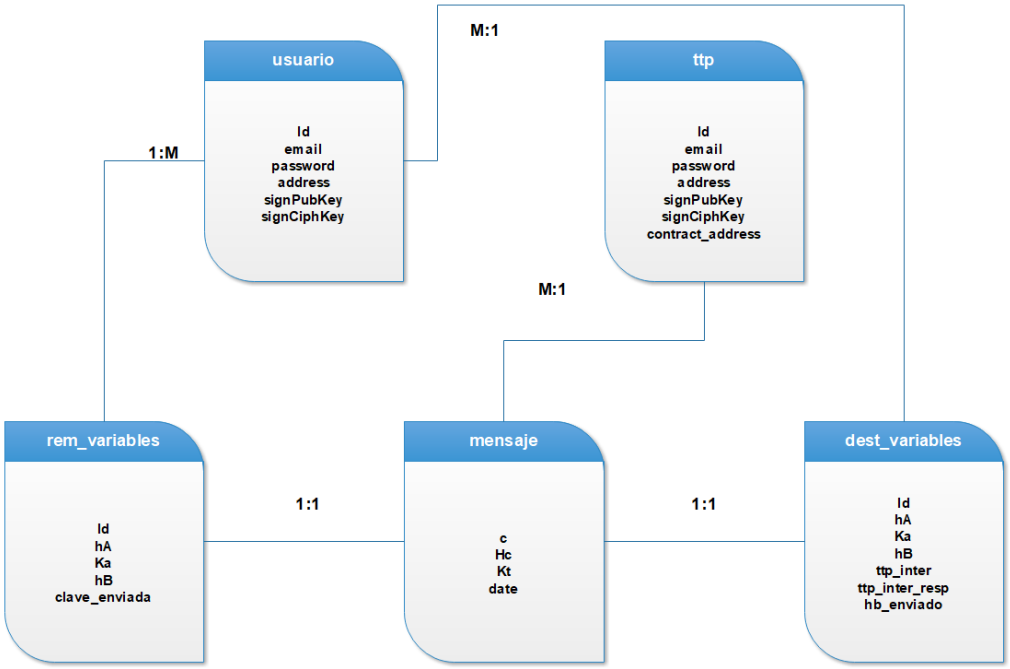

Figura 6.3: Diseño de la base de datos para el intercambio simple.

- **rem_variables.** Contiene todos los parámetros privados del intercambio a los que únicamente puede acceder el remitente: h_A enviado, k_A utilizada, h_B recibido y una variable booleana que indica si ya ha compartido la clave con el destinatario.

- **dest_variables.** Contiene todos los parámetros privados del intercambio a los que únicamente puede acceder el destinatario: h_A recibido, k_A recibido, h_B utilizado, justificante de solicitud de intervención de la TTP y su correspondiente respuesta en caso de que sea necesaria. Finalmente, cuenta también con una variable booleana que le indica si ya ha realizado el envío de la prueba de NRR.

6.4.2 Base de Datos para Notificaciones Certificadas Multiparte

Para el desarrollo de la aplicación de intercambio multiparte, la base de datos utilizada será prácticamente idéntica. Únicamente será necesario cambiar los siguientes parámetros:

- La relación entre *mensaje* y *dest_variables* se ha visto incrementada a una relación 1:M, puesto que cada mensaje tendrá múltiples destinatarios, y por tanto, múltiples parámetros que considerar.

- La relación entre *mensaje* y *rem_variables* se ha visto incrementada a una relación 1:M, puesto que el remitente recibirá las pruebas de NRR de cada uno de los usuarios del conjunto B.

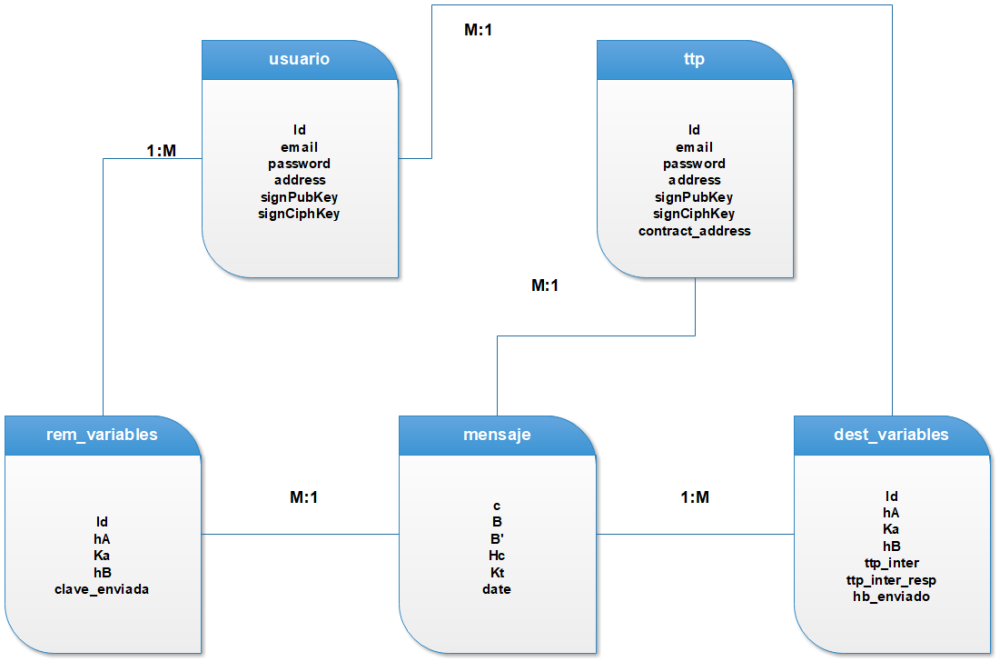

Figura 6.4: Diseño de la base de datos para el intercambio multiparte.

- En la tabla *mensaje* se han añadido dos campos: B (conjunto de usuarios que participan en el intercambio) y B'(conjunto de intercambios que han finalizado el intercambio).

6.5 Implementación del protocolo de Notificaciones Certificadas

La implementación del protocolo de intercambio entre dos partes se encontrará dividida en dos secciones. En la primera de ellas se mostrarán las dos versiones de los *Smart Contract* completos, pues son el objetivo principal del presente estudio. La segunda parte consistirá en la integración del contrato en una aplicación completamente funcional, en la que se mostrarán las funcionalidades más representativas.

6.5.1 Smart Contract sin Identificadores

La primera versión del *Smart Contract* realizada para el intercambio entre dos partes se ha realizado sin hacer uso de identificadores de mensajes. Esto significa que, un contrato será utilizado para únicamente un intercambio de notificaciones certificada. El código 6.1 muestra la lógica completa de este contrato.

Código 6.1: Smart Contract sin Identificadores

```
pragma solidity ^0.4.11;

contract CertifiedMail {

   //Parties involved
    address sender;
    address receiver;
    address ttp;

    string hB; //NRR proof
    string keyB; //Symmetric key encrypted with pubKey from B

    //Possible states
    enum State { created, cancelled, finished }
    State public state;
    bytes15 stateString;

    function CertifiedMail (address _sender, address _receiver){
        ttp = msg.sender;
        sender = _sender;
        receiver = _receiver;
        state = State.created;
        }


    event cancelEvent(
        string cancelResponse
        );

    event finishEvent(
        string resolveResponse
        );


    function cancel() {
         if(msg.sender==sender){
             if(state==State.created){
                 state=State.cancelled;
                 cancelEvent(getState());
             }else if (state == State.finished){
                 cancelEvent(hB);
             }
         }
    }

    function finish(string _hB, string _keyB){
        if (msg.sender==ttp){
            if(state==State.cancelled){
                finishEvent(getState());
            }else{
                hB=_hB;
                keyB=_keyB;
                state=State.finished;
                finishEvent(getState());
            }
        }
```

```
57      }
58
59      function getState() view public returns (string){
60          if (state==State.cancelled) return "Cancelled";
61          if (state==State.created) return "Created";
62          if (state==State.finished) return "Finished";
63      }
64  }
```

En este contrato se pueden encontrar tres funciones principales. La primera de ellas es el constructor. Este constructor será invocado durante el despliegue del contrato por parte de la TTP, y se encargará de definir la dirección pública de la *Wallet* de remitente y destinatario. De este modo se asegurará que ningún usuario externo al intercambio pueda acceder al resto de funciones.

La segunda función principal corresponde con el subprotocolo de cancelación definido en la sección 4.5.2. En primer lugar, antes de poder realizar alguna acción, debe comprobarse que la ejecución del método corresponde al remitente del mensaje, de lo contrario, la invocación de este método no dará resultado. Una vez realizada esta comprobación, se procederá a realizar el intento de cancelación. En el supuesto de que el intercambio no haya sido previamente finalizado, la cancelación se realizará de manera satisfactoria mediante el cambio de la variable de estado del contrato y la invocación de un evento que será recogido posteriormente por la aplicación web mediante el uso de *web3*. Si, por el contrario, el intercambio había sido finalizado anteriormente por el destinatario, se le entregará al remitente el correspondiente h_B como prueba de NRR también mediante la invocación de un evento.

Finalmente, la última función relevante del contrato corresponde con el subprotocolo de finalización definido en la sección 4.5.3. Como ya se ha expuesto en la fase de diseño, la interacción con el contrato será realizada por la TTP. Por lo tanto, la primera comprobación que se deberá realizar es si el método está siendo invocado por la tercera parte de confianza. De ser así se comprobará el estado actual del mensaje. En caso de estar cancelado, se avisará a la TTP de esta situación. Por otro lado, si el mensaje no ha sido cancelado previamente, el *Smart Contract* almacenará la prueba de NRR para proporcionársela al remitente posteriormente si es necesario. Es importante resaltar que, a pesar de que no se encuentra indicado en el diseño inicial, se ha decidido también almacenar un segundo parámetro llamado *keyB*, que corresponde con la clave simétrica k cifrada con la clave pública de B. Esta decisión de implementación ha sido tomada con la intención de proporcionar una confirmación adicional de que la clave ha sido puesta a disposición del remitente.

No obstante, a pesar de proporcionar las funciones especificadas en la fase de diseño del protocolo, este *Smart Contract* no es idóneo para el intercambio de notificaciones certificadas inicial. Tal y como se indica en la Figura 6.5 este contrato no proporciona la característica principal de *optimismo* en el intercambio. Esto se debe a que, al tener un contrato único para cada intercambio en el que se encuentran identificados remitente y destinatario, la TTP debe desplegar un contrato para cada uno de estos intercambios. Esto significa que, remitente y destinatario no cuentan con la posibilidad de realizar un intercambio completo sin intervención de la tercera parte. Por esta razón, se ha decidido que el contrato especificado en esta sección no va a ser el que se implemente en la versión final de la aplicación funcional.

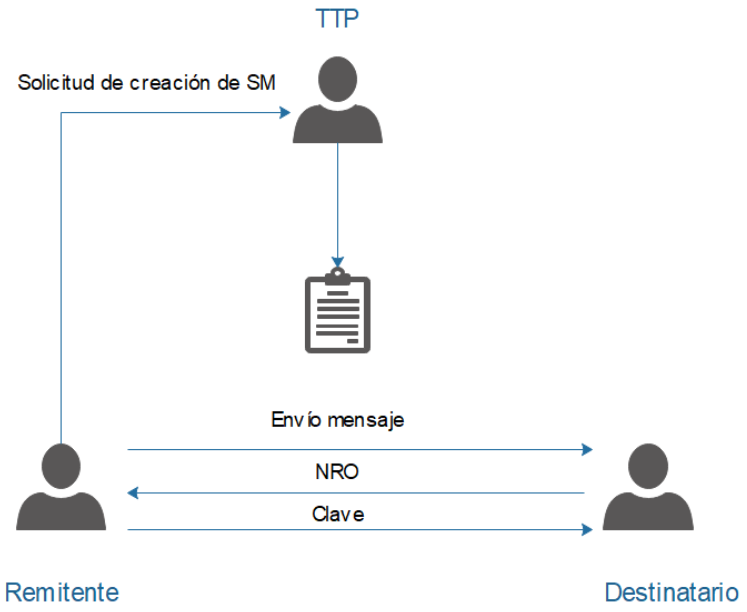

Figura 6.5: Intercambio con el Smart Contract sin Identificadores.

6.5.2 Smart Contract con Identificadores

Debido a las limitaciones derivadas de la implementación expuesta anteriormente, se ha desarrollado un *Smart Contract* alternativo que proporciona las características deseadas. A diferencia del contrato anterior, la versión nueva no identifica un único intercambio, sino que será utilizado por la TTP para gestionar múltiples intercambios entre distintos remitentes y destinatarios mediante el uso de identificadores de mensajes.

Código 6.2: Smart Contract con Identificadores

```
1  pragma solidity ^0.4.11;
2
3  contract IDCertifiedMail {
4
5     enum State {cancelled, finished }   //No need for created state.
6     address ttp;
7     struct Message{
8        address sender;
9        address receiver;
10       string hB;
11       string keyB;
12       State state;
13       bool isValue;
14    }
15
16    //Hash table with all messages
17    mapping(address => mapping(uint => Message)) messages;
18
19
```

```
20      constructor(){
21          ttp = msg.sender;
22      }
23
24      event cancelEvent(
25          string cancelResponse
26          );
27
28      event finishEvent(
29          string resolveResponse
30          );
31
32      //Main cancel protocol
33      function cancel(uint id,address receiver) {
34
35          if(messages[msg.sender][id].isValue){
36              if(messages[msg.sender][id].state==State.cancelled){
37                  cancelEvent(stateToString(messages[msg.sender][id].state));
38              }else{
39                  cancelEvent(messages[msg.sender][id].hB);
40              }
41          }else{
42            createMessage(id,msg.sender,receiver, "",
43                          "",State.cancelled);
44          }
45      }
46
47      //Main finish protocol
48      function finish(uint id,address sender,address receiver,
49                          string _hB, string _keyB){
50
51          if(msg.sender==ttp){
52              if(messages[sender][id].isValue){
53                    finishEvent(stateToString(messages[sender][id].state));
54              }else{
55                  createMessage(id,sender,receiver, _hB,
56                              _keyB,State.finished);
57              }
58          }
59      }
60
61       //Private function to create message
62       function createMessage(uint id, address sender, address receiver,
63                           string _hB,string _keyB,State state) private {
64
65              messages[sender][id].sender=sender;
66              messages[sender][id].receiver=receiver;
67              messages[sender][id].hB=_hB;
68              messages[sender][id].keyB=_keyB;
69              messages[sender][id].state=state;
70              messages[sender][id].isValue=true;
71       }
72
73
74       function stateToString(State state) view private returns (string){
75           if (state==State.cancelled) return "Cancelled";
76           if (state==State.finished) return "Finished";
```

```
77      }
78  }
```

A pesar de proporcionar unas funcionalidades muy similares al contrato anterior, el código presenta algunas diferencias notables. En primer lugar se encuentra un *struct* que contiene los parámetros identificativos de un intercambio. Esto se debe a que, en este caso, el contrato permite la gestión de múltiples mensajes, y por tanto, es necesario el uso de variables que permitan almacenar el contenido de cada uno de ellos. Estas estructuras de datos serán almacenadas en *maps*, el equivalente en *Solidity* a las tablas de Hash.

Por motivos de seguridad, se ha determinado que será necesario crear un *map of maps*. Esta medida se debe a que en el presente código, remitente y destinatario no son determinados en la creación del contrato (en el constructor), sino en la creación del mensaje (durante la cancelación y finalización). De este modo, no es posible comprobar que la cancelación de un mensaje con un *Id* determinado esté siendo realizada por el remitente del mensaje. Esto se debe a que un atacante puede haber descubierto el identificador, y hacerse pasar por el remitente para así cancelarlo. Con el objetivo de resolver esta problemática, se hará uso de un *map of maps*, de modo que cada mensaje se encuentre definido por dos claves, un identificador del mensaje y la dirección pública del remitente. De este modo, un atacante nunca podrá realizar la cancelación de un mensaje que no le pertenece. En la Figura 6.6 se muestra de manera esquemática la resolución de este problema.

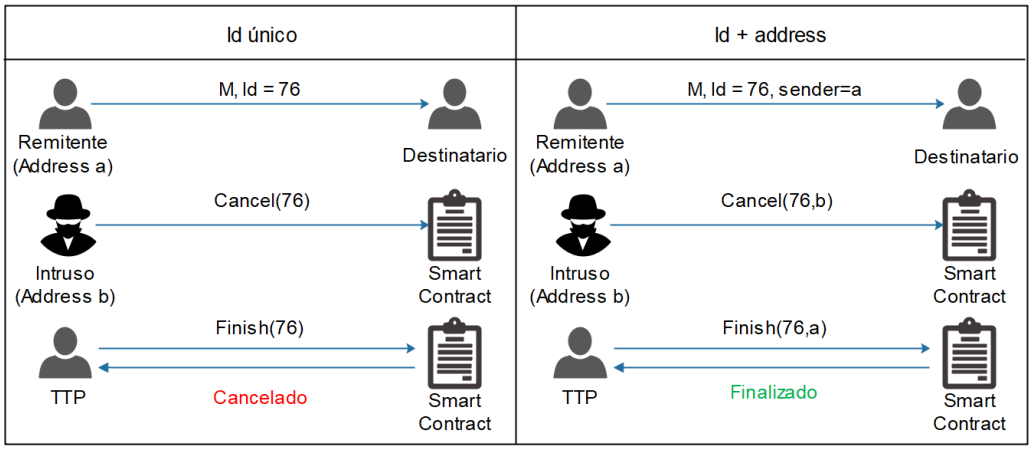

Figura 6.6: Seguridad del uso de *map of maps*.

De manera similar al contrato anterior, cada mensaje contiene un parámetro identificativo que indica el estado en el que se encuentra el intercambio. El valor de esta variable puede ser modificado mediante las invocaciones de los métodos *cancel()* y *finish()*.

La lógica del método de cancelación presenta una similitud importante respecto de la versión anterior. En este caso, en primer lugar se comprueba si ya existe un mensaje correspondiente a ese identificador y remitente. De ya existir, significa que o bien el intercambio ya ha sido cancelado o, en su defecto, ya ha sido finalizado. De encontrarse

cancelado, se indica al usuario el estado actual, mientras que de lo contrario, se le proporciona la prueba de NRR (h_B). Asimismo, si el mensaje no existe, significará que no se ha realizado una cancelación o finalización anteriormente, por lo que es necesario crear el mensaje correspondiente con el estado del intercambio cancelado (6.6).

La función de finalización también deberá ser ejecutada por la TTP. Del mismo modo que en el método de cancelación, la primera comprobación a realizar es si el mensaje ya ha sido creado. Si esta comprobación es afirmativa, significa también que el intercambio ya ha sido cancelado o finalizado anteriormente, por lo que únicamente será necesario indicar a la TTP el estado actual. Por otro lado, en caso de no estar creado, se deberá crear el mensaje a través del identificador y la dirección del remitente, y adicionalmente se deberán añadir los parámetros h_B y k cifrada con la clave pública de B. Finalmente, se indicará que el mensaje se encuentra finalizado.

De este modo, ha quedado demostrado que el uso de identificadores para el *Smart Contract* de intercambio entre dos partes es necesario. Puesto que, de este modo, es posible mantener la característica de principal de un intercambio *optimista* sin tener que introducir pasos adicionales en el intercambio.

6.6 Implementación del protocolo de Notificaciones Certificadas Multiparte

Esta etapa de la implementación consistirá en el desarrollo del protocolo especificado en el capítulo 5. En esta ocasión se ha desarrollado un único *Smart Contract*, el cual se encuentra directamente basado en el expuesto en la sección 6.5.2, con la posibilidad adicional de que un mismo mensaje tenga asociados múltiples destinatarios.

Código 6.3: Smart Contract Multiparte

```
1  pragma solidity ^0.4.11;
2
3  contract Multiparty {
4
5     //Variables
6
7     enum State {cancelled, finished}
8     address ttp;
9     struct Receiver{
10       address receiver;
11       string hB;
12       string keyB;
13       State state;
14       bool isValue;
15    }
16    struct Message{
17       uint id;
18       address sender;
19       mapping(address => Receiver) receivers; //All receivers
20       bool isValue;
21    }
22    mapping(address => mapping(uint => Message)) messages;
23
24    //Constructor
25
```

```
26      constructor(){
27          ttp = msg.sender;
28      }
29      //Events
30
31      event cancelEvent(
32          address cancelledUser,
33          string cancelResponse
34          );
35
36      event finishEvent(
37          string resolveResponse
38      );
39
40      //Main functions
41
42      function cancel (uint id,address[] cancelledReceivers) {
43
44          if (messages[msg.sender][id].isValue){
45
46              for (uint i = 0; i<cancelledReceivers.length;i++){
47
48                  address receiverToCancel = cancelledReceivers[i];
49
50                  if ((messages[msg.sender][id].receivers[receiverToCancel].
51                      isValue)&&(messages[msg.sender][id].receivers
52                      [receiverToCancel].state==State.finished)){
53
54                      cancelEvent(receiverToCancel,messages[msg.sender]
55                      [id].receivers[receiverToCancel].hB);
56
57                  } else if (!messages[msg.sender][id].receivers
58                              [receiverToCancel].isValue){
59
60                       addReceiver(id,msg.sender,receiverToCancel,
61                                   "","",State.cancelled);
62                       cancelEvent(receiverToCancel,stateToString(messages
63                                  [msg.sender][id].receivers[receiverToCancel].
64                                  state));
65
66                  } else {
67                      cancelEvent(receiverToCancel,stateToString(messages
68                      [msg.sender][id].receivers[receiverToCancel].state));
69                  }
70
71              }
72
73          } else {
74
75              createMessage(id,msg.sender);
76
77              for(uint j= 0; j<cancelledReceivers.length;j++){
78
79                  address receiverToAdd = cancelledReceivers[j];
80
81                  addReceiver(id,msg.sender,receiverToAdd,"","",
82                              State.cancelled);
```

```
83
84              cancelEvent(receiverToAdd,stateToString(messages[msg.sender]
85              [id].receivers[receiverToAdd].state));
86          }
87      }
88  }
89
90
91  function finish(uint id,address sender,address receiver,string _hB,
92                  string _keyB){
93
94      if (msg.sender==ttp){
95
96          if (messages[sender][id].isValue){
97
98              if((messages[sender][id].receivers[receiver].isValue)
99                &&(messages[sender][id].receivers[receiver].
100               state==State.cancelled)){
101
102                 finishEvent(stateToString(messages[sender][id].
103                             receivers[receiver].state));
104
105             }else if(!messages[sender][id].receivers[receiver].isValue){
106
107                 addReceiver(id,sender,receiver,_hB,_keyB,State.finished);
108
109                 finishEvent(stateToString(messages[sender][id].
110                             receivers[receiver].state));
111             }else{
112                 finishEvent(stateToString(messages[sender]
113                             [id].receivers[receiver].state));
114             }
115
116         }else{
117
118             createMessage(id,sender);
119
120             addReceiver(id,sender,receiver,_hB,_keyB,State.finished);
121
122             finishEvent(stateToString(messages[sender][id].
123                         receivers[receiver].state));
124         }
125     }
126 }
127
128 function createMessage(uint id, address sender) private {
129     messages[sender][id].id=id;
130     messages[sender][id].sender=sender;
131     messages[sender][id].isValue=true;
132 }
133
134 function addReceiver(uint id, address sender, address receiver,
135                 string _hB, string _keyB, State _state) private{
136
137     messages[sender][id].receivers[receiver].receiver=receiver;
138     messages[sender][id].receivers[receiver].hB=_hB;
139     messages[sender][id].receivers[receiver].keyB=_keyB;
```

```
140         messages[sender][id].receivers[receiver].state=_state;
141         messages[sender][id].receivers[receiver].isValue=true;
142     }
143
144     function stateToString(State state) view private returns
145                     (string){
146
147         if (state==State.cancelled) return "Cancelled";
148         if (state==State.finished) return "Finished";
149     }
150 }
```

La primera modificación que se debe realizar, para permitir la asociación de múltiples destinatarios a un mismo mensaje es la creación de una nueva estructura de datos para los destinatarios. Esta estructura de datos será la que se encargue de almacenar los parámetros necesarios y determinar el estado en el que se encuentra el intercambio. De este modo, el estado del intercambio ya no queda definido de manera única en el mensaje, sino que depende de cada destinatario. El conjunto de destinatarios se almacenará dentro del *struct* mensaje mediante el uso de un *map*.

Una vez realizada esta modificación es importante considerar los cambios que esto supone sobre las funciones *cancelar()* y *finalizar()*.La cancelación sigue el mismo principio que el contrato expuesto en 6.5.2, con la principal diferencia de que esta vez el parámetro de entrada del método no es un único destinatario, sino que es un *array* que contendrá el conjunto B". La función se encarga de recorrer este conjunto, y cancelar el intercambio para cada uno, o en su defecto, retornar el correspondiente h_B.

La finalización, por su parte, es la que menos modificaciones ha recibido. El funcionamiento es exactamente idéntico al expuesto en la sección 6.5.2, con la única diferencia de que se debe comprobar si B_i se encuentra ya dentro de $B" - cancelado$. De ser así se avisa a la TTP de esta situación. De lo contrario, se incluye a B_i en $B" - finalizado$ y se almacena la prueba de NRR h_{Bi} y la clave k cifrada con su correspondiente clave privada.

6.7 Desarrollo de la aplicación funcional

Una vez vistos los contratos correspondientes a los distintos protocolos se procederá a mostrar la realización de la aplicación web sobre la que se integrarán. Es importante mencionar que la aplicación solo se realizará para el Smart Contract multiparte al tratarse del protocolo más completo de los presentados anteriormente. Tal y como se ha visto en la sección 6.2, la aplicación no consiste en una *DApp* clásica. Esto se debe a que estas aplicaciones distribuidas se caracterizan por ejecutar toda la parte de *backend* sobre una red P2P descentralizada. Sin embargo, esta aplicación web funciona de manera híbrida, puesto que algunas de las funcionalidades son proporcionadas por un servidor central y otras por la blockchain.

La implementación se ha realizado como resultado de un desarrollo basado en funcionalidades y se ha proporcionado una interfaz gráfica para su interacción con los usuarios, tal y como puede verse en las Figuras 6.7 y 6.8. El código completo de la aplicación puede encontrarse en el repositorio de *github* llamado SECOMUIB [1]. A

[1] https://github.com/secomuib/MultipartyConfidentialCertifiedNotifications.git

Figura 6.7: Interfaz Gráfica. Mensajes enviados.

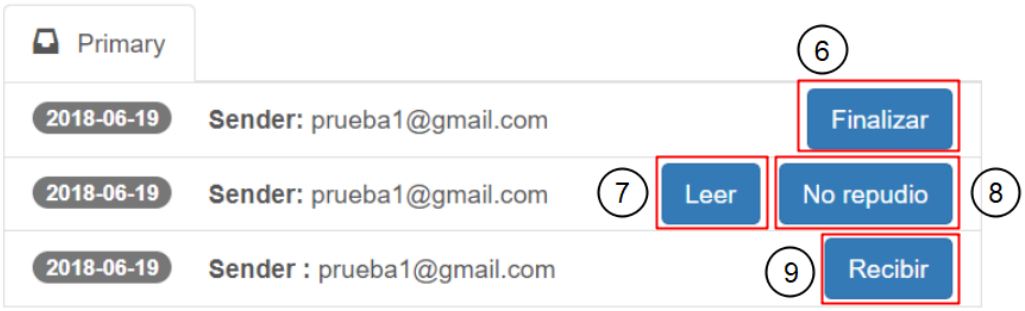

Figura 6.8: Interfaz Gráfica. Mensajes recibidos.

continuación se detallarán las funcionalidades principales a través de las imágenes proporcionadas.

- **Registro de Usuarios.** El registro de usuarios se ha realizado de acuerdo a los parámetros establecidos en el diseño del protocolo. Los usuarios deben contar con un nombre de usuario, una contraseña y una *Wallet* de Ethereum disponible desde *Metamask* para poder registrarse. Una vez registrada, a través de la librería SJCL se generarán las correspondientes parejas de claves públicas y privadas para el cifrado y firma de mensajes. Por motivos de simplicidad y de seguridad, se ha decidido que las claves privadas serán almacenadas en el navegador personal de cada usuario. Las claves públicas por su parte, se almacenarán en la base de datos del servidor principal.

- **Despliegue de contratos**. Una de las funcionalidades esenciales en la aplicación es la posibilidad de desplegar contratos sobre la blockchain de manera automatizada. Este proceso se lleva a cabo durante el registro de las terceras partes.

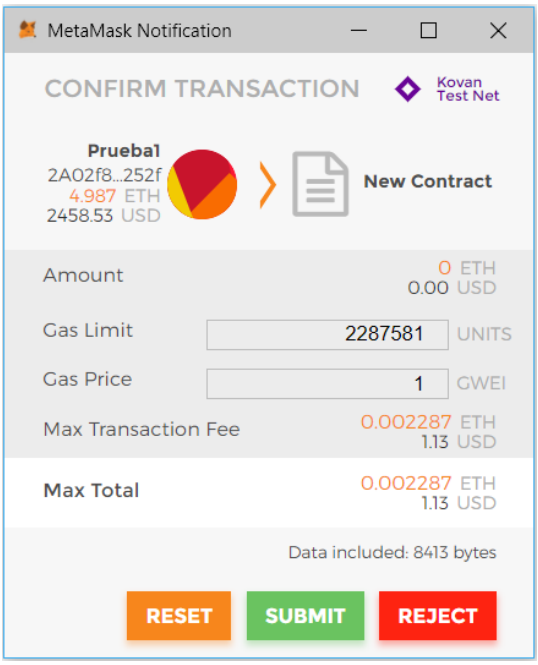

Figura 6.9: Notificación de transacción. Metamask.

Para el despliegue del contrato son necesarios dos parámetros identificativos de este: la Application Binary Interface (ABI) y el *Bytecode* [19]. La ABI se encarga de definir como se realizarán las comunicaciones con el contrato. El *bytecode* por su parte contiene el código del contrato de tal manera que la EVM sea capaz de entenderlo. No obstante, estos dos parámetros se obtienen a partir de la compilación del contrato. Como ya se ha explicado en la sección 6.1.1, la compilación automática se considera fuera del alcance del proyecto. Por lo tanto, la obtención de estos dos parámetros se realizará a través de *Remix*.

Una vez obtenidos los parámetros, se deberán introducir en la correspondiente función de Javascript. Esta función se ejecutará una vez finalizado el registro de la TTP y mediante el uso de *Metamask* se le solicitará que acepte la transacción. Esta solicitud puede verse en la Figura 6.9. En la imagen aparecen distintos campos. El primero, *amount* indica la cantidad a transferir. Al tratarse de la ejecución de un contrato que no requiere transferencias, el valor de este campo es 0. A continuación aparece el parámetro *gas limit*, encargado de determinar el valor máximo de gas que va a utilizarse. Por su parte, *gas price* determina el coste (en Gweis) de cada unidad de gas. De este modo, teniendo en cuenta estos parámetros, se calcula el coste máximo de la transacción, aunque el coste real puede encontrarse por debajo de este valor.

En caso de aceptar la solicitud y de que se cuenten con fondos suficientes para realizar la transacción, el contrato será desplegado. La respuesta de esta función indicará la dirección pública en la que se encontrará el contrato, que será almacenada para ponerse en disposición de los distintos usuarios que requieran su uso.

- **Realización del intercambio.** El envío de notificaciones es la funcionalidad principal de la aplicación. Como ya se ha visto a lo largo del documento, se encuentra compuesto por tres fases.

  La primera se iniciará con la redacción del mensaje por parte de un usuario (Figura 6.7-**1**). Para ello será imprescindible el uso de la librería SJCL. De este modo, mediante Javascript será posible realizar el cifrado y firma de los parámetros necesarios para el intercambio (c, k_T y h_A). Estas variables no se distribuirán de manera directa entre emisor y destinatario(s), puesto que se trata de una arquitectura cliente-servidor, estos parámetros serán enviados al servidor, procesados y almacenados en la base de datos correspondiente.

  La segunda fase tendrá lugar con la recepción del mensaje por parte del destinatario(Figura 6.8-**9**). En este punto, el usuario podrá confirmar la recepción del mensaje. De así hacerlo, obtendrá los parámetros $H(c), k_T$ e Id de la base de datos, y mediante su clave privada almacenada en el navegador, realizará la firma de estos para obtener h_B. Del mismo modo, el envío de h_B tampoco se llevará a cabo de manera directa, sino mediante el servidor y la base de datos.

  Finalmente, cuándo el remitente reciba la prueba de NRR (Figura 6.7-**4**), procederá a enviar la clave simétrica firmada k_A al destinatario (Figura 6.7-**5**). El envío será llevado a cabo únicamente si la prueba de NRR es correcta. Es importante notar que esta clave ha sido almacenada anteriormente durante la fase inicial del intercambio. No obstante, hasta este preciso momento se encuentra almacenada en la base de datos de tal manera que únicamente el remitente tiene acceso a ella.

- **Lectura mensajes.** La base de datos no contiene el mensaje descifrado. Por lo tanto, cada vez que un usuario desee leer un mensaje, se realiza una petición al servidor para que proporcione el texto cifrado c y la clave k_A (Figura 6.8-**7**). Una vez obtenidos estos parámetros, mediante el uso de Javascript y la librería SJCL se realizará el descifrado del mensaje y se le mostrará el resultado al usuario. Del mismo modo, una vez el destinatario haya obtenido la clave para descifrar el mensaje, también podrá obtener las dos partes de la prueba de NRO (Figura 6.8-**8**).

- **Cancelar intercambio.** En las explicaciones previas se ha podido ver como la cancelación se lleva a cabo a través del *Smart Contract*. No obstante, el usuario debe poder interactuar directamente con el contrato desde la aplicación web en caso de no recibir la prueba de NRR (Figura 6.7-**3**). Para ello, deberá obtener la dirección del contrato perteneciente a la TTP implicada en el intercambio. Una vez obtenida esta dirección a través de la base de datos, el usuario interactuará con el contrato mediante el uso de Javascript y web3.js. Del mismo modo que en el despliegue del contrato, serán necesarios el *bytecode* y la *ABI* para poder establecer la comunicación. Finalmente, al solicitar la cancelación aparecerá la notificación de *Metamask* de que debe confirmar la transacción. En caso de tener *Ether* suficiente para hacer frente al precio del *gas* necesario, procederá a realizarse la cancelación.

Una vez aceptada la transacción, se deberá esperar a la correspondiente invocación del *evento* de cancelación por parte del Smart Contract. Los eventos establecen la comunicación entre la blockchain y las *DApps*, por ello, la aplicación web se encontrará a la espera de este evento, y una vez lanzado, obtendrá información sobre la ejecución del método de cancelación (h_B o confirmación de cancelación) y lo almacenará en la base de datos.

- **Solicitar finalización.** La solicitud de finalización la llevará a cabo un destinatario que haya enviado el correspondiente h_{Bi} pero no haya recibido la clave k_A. Debido a que la interacción con el contrato para la finalización debe llevarla a cabo la TTP, el remitente deberá solicitar su intervención (Figura 6.8-**6**). Para ello, mediante javascript y la librería SJCL creará la solicitud de intervención y la depositará en la base de datos. De este modo, la TTP podrá proceder a la finalización.

- **Finalizar intercambio.** La finalización del intercambio, tal y como ya ha sido expuesto, da inicio con la solicitud de finalización de un remitente. Al recibir esta solicitud la TTP procederá a realizar la finalización a través de una interfaz gráfica prácticamente idéntica a la del resto de usuarios. Al recibir los parámetros expuestos en 5.4.3, la TTP deberá, en primer lugar, comprobar la veracidad de estos. De ser correctos, descifrará la clave k_T que posee y la cifrará con la clave privada de B_i. En ese entonces, accederá al contrato intentando dejar constancia de h_{Bi} y la clave.

  Una vez ejecutada la finalización, al igual que en la cancelación, deberá esperar hasta la recepción de la respuesta mediante el evento de finalización. De obtener una respuesta satisfactoria firmará la clave k y la enviará a B. De lo contrario, deberá presentar un justificante a B de que la finalización se ha intentado realizar pero el intercambio se encuentra cancelado ($PR_T[H("cancelled"), h_{Bi}]$).

6.8 Sumario de la implementación

El objetivo de este capítulo ha sido mostrar el proceso completo de desarrollo de los *Smart Contracts* correspondientes a los dos protocolos especificados así como de la aplicación sobre la que se integran. En primer lugar, se han expuesto las tecnologías, arquitectura y bases de datos necesarias para poder permitir el desarrollo de una aplicación de notificaciones certificadas mediante *Smart Contracts*. Una vez sentadas las bases, se ha procedido a mostrar y explicar la parte más importante de la implementación: el desarrollo del contrato inteligente.

En el caso del intercambio simple, se ha intentado la implementación del diseño por medio de dos contratos distintos. El primer contrato expuesto, el cual no hacía uso de identificadores, no ha arrojado resultados satisfactorios. Estos resultados se deben a que, a pesar de permitir el intercambio de notificaciones confidenciales, el intercambio dejaba de ser *optimista*, puesto que se requería la intervención de la TTP para hacer el despliegue antes de iniciar cada intercambio. Para solventar esta situación, se ha presentado un segundo contrato que, mediante el uso de identificadores, ha permitido obtener los resultados esperados y la gestión de múltiples intercambios.

Una vez realizados los contratos del intercambio entre dos partes, se ha desarrollado el contrato perteneciente al intercambio multiparte. Este se ha fundamentado en el *Smart Contract* con identificadores anterior, y mediante ciertas ampliaciones, ha permitido inclusión de múltiples destinatarios en un mismo mensaje.

Finalmente, se ha expuesto la implementación de la aplicación completa en la que se integraban todas las funcionalidades requeridas en la fase de diseño. Una vez realizada la implementación, en el siguiente capítulo se discutirán los resultados obtenidos durante esta fase y se comprobará la validez de la solución aportada.

CAPÍTULO 7

ANÁLISIS DE PROPIEDADES Y RENDIMIENTO

En este capítulo se presentarán y discutirán los resultados obtenidos durante la etapa de implementación. Las pruebas se basarán en dos partes fundamentales. La primera de ellas consiste en un análisis exhaustivo del protocolo, mientras que la segunda se basa en el rendimiento de la implementación.

7.1 Análisis del protocolo

El objetivo de esta primera etapa es determinar las características que presenta la aplicación desarrollada. Es importante tener en cuenta que este análisis se ha realizado únicamente sobre la versión final de la aplicación, es decir, sobre el protocolo de intercambio multiparte. No obstante, por motivos de similitud, los resultados obtenidos en la aplicación de intercambio multiparte serán extrapolables a los del intercambio simple.

7.1.1 Análisis de propiedades

A continuación se analizarán las propiedades del protocolo presentadas en 2.1.3 [1]. Para la demostración de algunas de las propiedades se utilizarán imágenes obtenidas de la aplicación.

Propiedad 1: *El protocolo es eficaz.*

Demostración: Si A y todos los B_i actúan de manera lícita, se seguirá el subprotocolo de intercambio. Este protocolo permitirá el intercambio de parámetros entre las distintas partes sin necesidad de intervención de la TTP.

El intercambio se inicia con la redacción del mensaje por parte del remitente, tal y como se muestra en la Figura 7.1. Una vez decida enviarlo, corresponderá con la primera fase del intercambio, en la que todos los destinatarios reciben los parámetros c, k_T, B y h_A.

Es importante aclarar que, a pesar de que la Figura 7.1 muestra el uso de direcciones de correo electrónico como identificadores de los usuarios, el uso de este parámetro

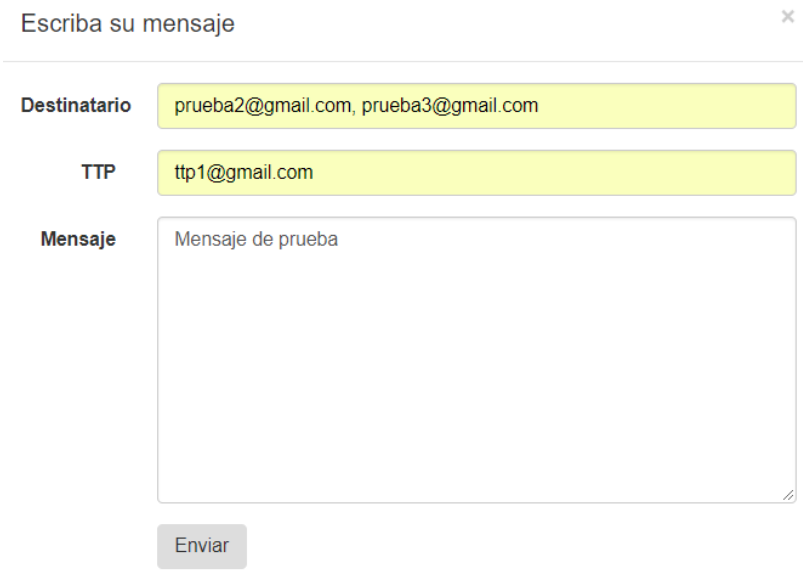

Figura 7.1: Redacción del mensaje

como identificador es totalmente arbitrario y no corresponde al envío de un correo electrónico. Es importante mencionar también que, a pesar de que cada usuario se encuentra identificado también por una dirección de la wallet Ethereum, esta no se ha utilizado como identificador principal del usuario. Esta decisión ha sido tomada con la intención de facilitar el modo de empleo a los distintos usuarios finales, evitando así el uso de largos y complejos identificadores.

Una vez enviada la notificación, los distintos usuarios no pueden aún proceder a la lectura de esta, puesto que, aún no poseen la clave para el descifrado. Por ello, primero deben enviar el correspondiente h_{Bi}. El envío de este parámetro se llevará a cabo cuando el usuario confirme la recepción del mensaje, tal y como puede apreciarse en la figura 7.2.

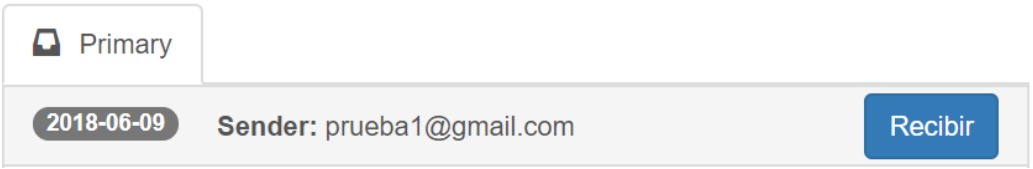

Figura 7.2: Envío de la prueba de NRR.

Una vez enviado el parámetro h_{Bi}, el remitente podrá proceder a enviar la correspondiente clave k_A. Tal y como vemos en la Figura 7.3 aunque haya usuarios que no sean honestos, el intercambio se podrá finalizar de manera correcta con aquellos que sí lo sean.

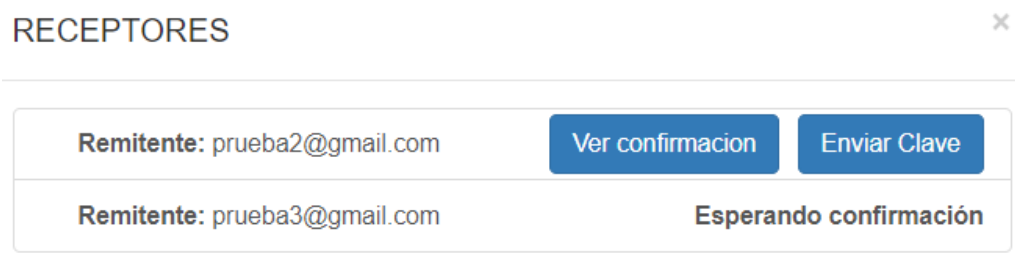

Figura 7.3: Envío de la clave a B_i

Finalmente, el remitente puede proceder a la lectura del mensaje, dando así por finalizado el intercambio y garantizando la propiedad de eficacia.

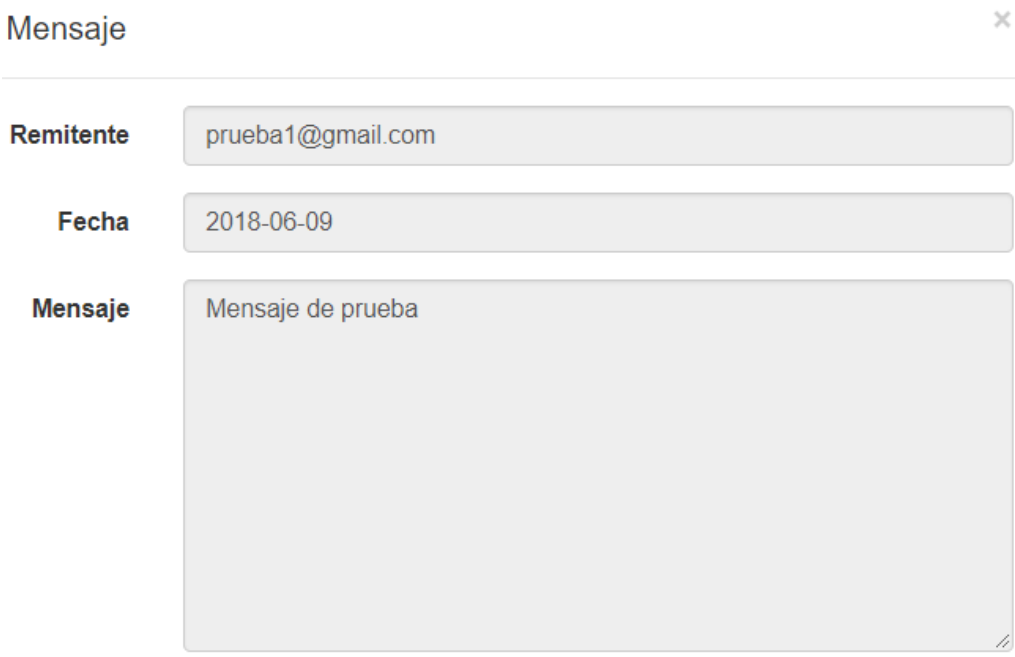

Figura 7.4: Mensaje descifrado por B_i.

Propiedad 2. *El protocolo es optimista*

Demostración: Tal y como se ha visto en la demostración de la propiedad 1. Es posible realizar un intercambio completo sin necesidad de que intervenga la TTP si los implicados actúan de manera honesta.

Propiedad 3. *El protocolo es eficiente*

Demostración: En las dos primeras propiedades se ha podido observar como la implementación permite el intercambio completo en tan solo tres pasos. De acuerdo con lo que se establece en [8], no es posible la realización de un intercambio equitativo en un número de pasos menor a tres. Por lo tanto, es posible afirmar que el protocolo cumple con el requisito de eficiencia, puesto que se realiza en el menor número de etapas posibles.

Propiedad 4. *El protocolo es equitativo*

Demostración. El protocolo actual presenta una equidad *débil*. Para demostrar esta situación debemos considerar dos casos distintos.

En primer lugar, el primer caso a considerar es que A no ha recibido el parámetro h_{Bi} después del paso 1 del intercambio. En este caso, A puede realizar la cancelación del intercambio. De hacerlo, existen dos escenarios posibles, ambos con un final equitativo. En el caso de que el destinatario ya haya solicitado la finalización del intercambio, el remitente obtendrá la prueba de NRR. De lo contrario, la cancelación del mensaje será satisfactoria y esto impedirá que el destinatario pueda proceder a la lectura del mensaje (Figura 7.5). En ambos casos se garantiza la equidad del intercambio.

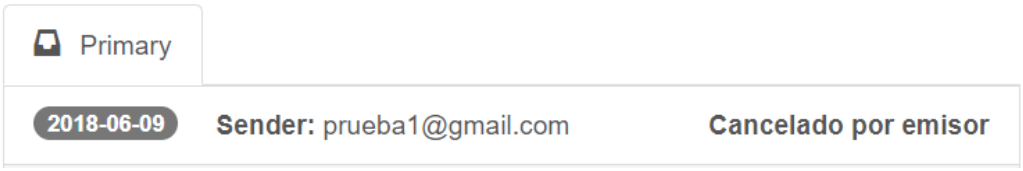

Figura 7.5: Mensaje cancelado por el remitente

La otra opción puede surgir cuando B_i ha realizado el envío de su correspondiente h_{Bi} pero no ha recibido k_A. En este caso, B puede acceder a la TTP y solicitar la finalización del intercambio. Si B demuestra satisfactoriamente ser el destinatario del mensaje M y el intercambio no se encuentra cancelado, la TTP le proporcionará la clave k'_T. De este modo el intercambio finalizará de manera equitativa.

No obstante, es posible afirmar que la equidad es *débil* a causa de un escenario en el que A puede obtener pruebas contradictorias. Este escenario puede darse cuando, habiendo recibido h_B, A accede posteriormente a la *blockchain* y cancela el intercambio. De este modo, A contaría con la prueba de NRR y al mismo tiempo con una evidencia en el *Smart Contract* de cancelación, dando lugar así a pruebas contradictorias e impidiendo obtener dos de las propiedades fundamentales: *equidad fuerte* y *transferibilidad de pruebas*.

Propiedad 5. *El protocolo es asíncrono.*

Demostración: La asincronía del protocolo es demostrable debido a que el remitente puede realizar la cancelación a través del *Smart Contact* y el destinatario la finalización a través de la TTP en cualquier momento del intercambio. El único factor temporal que cabe tener en cuenta es el tiempo que tardará la blockchain en confirmar la transacción [1].

Propiedad 6. *No repudio en origen*

Demostración: Al finalizar el intercambio, el destinatario siempre contará con las dos partes de la prueba de NRO. La primera parte la obtendrá en la primera fase del intercambio, mediante la recepción del parámetro h_A. Sin embargo, será necesaria una segunda parte de la prueba para completar la evidencia de NRO. Esta segunda parte podrá obtenerla de dos maneras distintas. Si el intercambio se produce de manera normal, A enviará k_A, que actuará como segunda parte de la evidencia. De lo contrario, si es necesaria la intervención de la TTP, la segunda parte de la prueba será k'_T. En cualquiera de los dos casos, tal y como se muestra en la Figura 7.6, al finalizar el intercambio A no podrá negar el envío del mensaje.

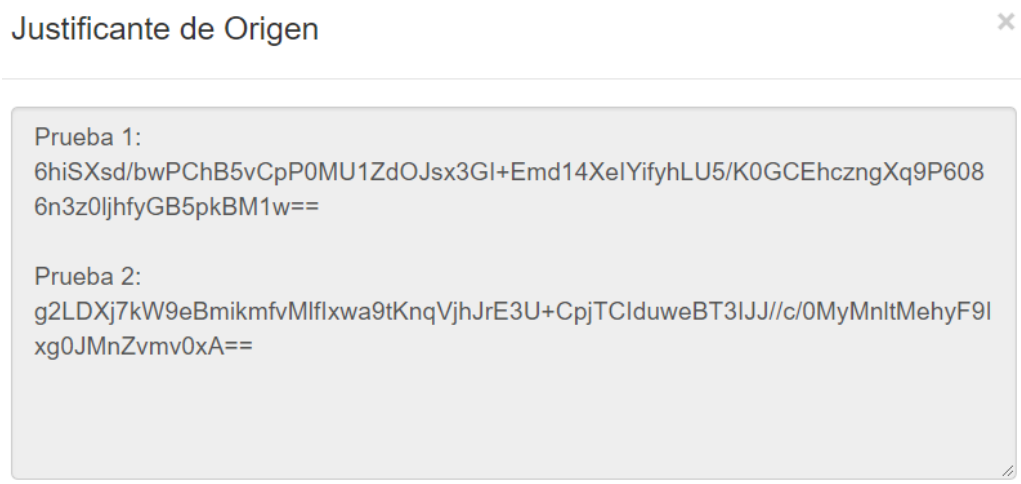

Figura 7.6: Pruebas de NRO recibidas

Figura 7.7: Prueba de NRR recibida

Propiedad 7. *No repudio en recepción*

Demostración: En caso de finalizarse el intercambio, con o sin intervención de la TTP, el remitente del mensaje siempre contará con el parámetro h_{Bi}, es decir, contará con la prueba de NRR, y por lo tanto, B_i no podrá negar haber recibido el mensaje (Figura 7.6).

Propiedad 8. *La TTP no necesita mantener el estado del intercambio*

Demostración: Como ya se ha visto anteriormente, la única función que realiza la tercera parte es la finalización del intercambio. Para realizarlo, la TTP no necesita mantener información sobre el estado en el que se encuentra. Esto se debe a que esta información es almacenada por el *Smart Contract*. Por tanto, la TTP podrá intentar realizar la finalización sin asegurarse primero de cuál es el estado del intercambio, ya que este le será indicado por el contrato.

Propiedad 9. *El protocolo proporciona confidencialidad en el intercambio*

Demostración: El protocolo garantiza la confidencialidad del intercambio tanto si se ha necesitado la intervención de la TTP como si no ha sido necesario. Esto se debe a que el mensaje se encuentra cifrado, y por tanto, ningún usuario externo al intercambio tiene acceso al contenido de este. Además, se presupone que la TTP es una entidad de confianza que no proporcionará la clave del mensaje de manera irregular a los usuarios.

Propiedad 10. *La TTP es verificable*

Demostración: En el presente protocolo existen dos casos en los que la TTP puede actuar de manera incorrecta.

El primero de ellos consiste en que la TTP acceda al *Smart Contract* e intente realizar la finalización del intercambio para el usuario B_i sin que este lo haya solicitado. En este caso, el comportamiento incorrecto de la TTP quedará demostrado por dos parámetros. En primer lugar, el valor de h_{Bi} introducido en el contrato no será correcto puesto que B_i no se lo habrá proporcionado. Además, será demostrable que ha actuado de manera ilícita puesto que la TTP no contará con h_{BiT}, que corresponde con la solicitud de intervención.

El segundo caso de procedimiento incorrecto de la tercera parte puede darse si esta informa erróneamente de que el intercambio ya ha sido cancelado sin comprobar el estado del contrato. Este comportamiento erróneo puede solventarse gracias al uso de la evidencia de intervención $PR_T[H("cancelled"), h_{Bi}]$. Por tanto, B_i puede demostrar este comportamiento comparando la evidencia de intervención con el estado real mostrado en la blockchain.

7.1.2 Limitaciones del protocolo

A pesar de las características expuestas anteriormente, el protocolo presenta una serie de limitaciones que deben considerarse.

- **Limitaciones por uso de blockchain.** Una de las características principales de hacer uso de la tecnología *blockchain* es que existe un cierto coste económico. El problema principal viene causado por una asimetría en el coste entre remitente y destinatario. Esto se debe a que la cancelación supone un coste económico sobre el remitente, puesto que es él quien debe interactuar con el contrato. Además, es importante notar que como mayor sea el número de destinatarios, mayor será el coste. Por otro lado, la finalización no supone coste alguno sobre los destinatarios, ya que es ejecutada por la TTP. Esta problemática puede dar lugar a situaciones en las que, una de las dos partes deba invertir recursos monetarios para la obtención de los parámetros deseados, mientras que la otra no. Una posible solución a este problema podría ser el almacenamiento de fondos en el contrato por parte de la TTP. De este modo, cuando un remitente realizase la cancelación, el contrato le transferiría una cantidad de monedas equivalente al coste de ejecución total. Esto supondría un mayor coste para la TTP pero eliminaría esta problemática.

  La otra limitación principal es que posibles atacantes podrían utilizar el protocolo de manera fraudulenta, solicitando finalizaciones a la TTP de manera sistemática. De este modo, podrían conseguir agotar el *Ether* que posee la entidad y que posteriormente no pudiese asumir los costes computacionales de las

finalizaciones de mensajes lícitos. Esta limitación podría solventarse mediante el uso de *listas negras* que se encontrasen almacenadas en el contrato. De este modo, si un destinatario ha solicitado un número determinado de finalizaciones en un intervalo de tiempo, será colocado en la lista negra temporalmente. Mientras se encuentre en esta lista, la TTP no podrá realizar la finalización para ese destinatario, reduciendo así el coste de gas para cada solicitud de manera notable.

- **Limitaciones del diseño.** Tal y como se explica en [8], el protocolo presenta problemas en casos de confabulación entre alguna de las partes y la TTP. Esto se debe a que después del primer mensaje del intercambio, la TTP puede proporcionar la clave k a B_i sin dejar constancia. De este modo, posteriormente cuando A vaya a realizar la cancelación, B_i habrá tenido acceso al mensaje descifrado, pero A no contará con la prueba de NRR. Si bien es cierto que B_i no contará con la prueba de NRO, pero en la mayoría de casos de notaciones certificadas, el parámetro de no repudio en origen no es tan importante como tener acceso al contenido del mensaje.

7.1.3 Comparación con Protocolos sin Blockchain

Una vez establecidas las propiedades y limitaciones del protocolo se procederá a compararlo con el protocolo multiparte original establecido en [2]. La comparación entre protocolos se basará en tres aspectos fundamentales:

- **Resolución de disputas**. La resolución de disputas es el primer factor diferencial entre ambos protocolos. En el presente protocolo no es necesario que un arbitro externo interrogue a ambas partes para descubrir el estado final del intercambio. Es suficiente con que interrogue a una de las partes y corrobore su versión con la información de la blockchain. Anteriormente, en el protocolo sin blockchain, el árbitro externo necesitaba acudir a ambas partes para poder determinar como había finalizado el intercambio [1].

- **Intervención de la TTP**. En el protocolo actual se ha logrado reducir la intervención de la TTP en la resolución de disputas. Esto se debe a que las cancelaciones son llevadas a cabo por el remitente a través del *Smart Contract* sin necesitar que la TTP intervenga de manera activa en el intercambio. Esto lo diferencia del protocolo original, ya que en este la TTP debía intervenir tanto en las cancelaciones como en las finalizaciones.

- **Almacenamiento de estado**. Tal y como se ha explicado a lo largo del documento, el objetivo principal de utilizar la tecnología blockchain es permitir que la TTP no deba almacenar el estado del intercambio. De este modo, es el *Smart Contract* el que se encarga de almacenar el estado en el que se encuentra cada intercambio. Esta situación anteriormente no era posible, ya que como indica J. Ferrer et al. en *"Certified Electronic Mail: Properties Revisited"* [4] las propiedades de *asincronía* y *no almacenamiento de estado por la TTP*, bajo su definición clásica, eran incompatibles. Con la aparición de la blockchain, esta limitación ha sido resuelta e incorporada en el protocolo. Sin embargo, bajo la implementación

actual sobre una aplicación web el protocolo no puede prescindir de una base de datos centralizada, pues es necesaria para realizar el intercambio. No obstante, la base de datos no proporciona información relativa al estado del intercambio a la TTP para realizar el proceso de cancelación o finalización, sino que es el Smart Contract quién proporciona esta información. En el caso de haber realizado la integración del contrato sobre una aplicación de escritorio ad hoc, la base de datos centralizada no sería necesaria para el intercambio.

7.2 Análisis de rendimiento

Uno de los factores claves a la hora de desarrollar tecnologías basadas en blockchain es el análisis de rendimiento. La importancia de este aspecto se debe a que en estas tecnologías el código no es ejecutado sobre la máquina local, sino sobre una red P2P distribuida. Esto significa que el coste computacional de ejecución de un contrato se traduce en un coste económico determinado y un cierto tiempo de espera. A continuación se valorarán estos dos parámetros para cada una de las funcionalidades de los tres contratos expuestos anteriormente.

Es importante establecer en primer lugar, que estos parámetros se han estudiado de acuerdo a las pruebas establecidas sobre la red *Kovan*. Esta red no es más que una red pública de prueba para Ethereum, dedicada especificamente para desarrolladores. Es por ello que los resultados obtenidos en esta sección, especialmente los resultados temporales, deben considerarse orientativos y servir únicamente para establecer comparaciones entre los contratos implementados y, bajo ningún concepto, deben considerarse como resultados absolutos en la blockchain principal de Ethereum.

7.2.1 Tiempo de espera

En primer lugar se evaluará el tiempo promedio que tarda en computarse cada una de las distintas funcionalidades del protocolo. Como ya se ha expuesto anteriormente, esto se realizará en la red de pruebas Kovan. De acuerdo con [20], esta red realiza la publicación de un nuevo bloque cada 4 segundos, por lo tanto los resultados van a encontrarse bastante alejados de un escenario real como el de *Ethereum* cuya publicación de bloques se realiza cada 16 segundos.

El tiempo de espera se ha calculado como la diferencia entre el tiempo de publicación del bloque correspondiente y la entrada de la transacción en la red P2P. Para ello se ha hecho uso de la herramienta *etherscan* [21]. Adicionalmente, cada acción se ha realizado un total de 15 veces con el objetivo de proporcionar la media aritmética de estos resultados.

Tal y como se puede apreciar en la tabla 7.1, los resultados temporales no son concluyentes, puesto que los resultados arrojados son prácticamente idénticos para todos los métodos y no corresponden con la cantidad y complejidad de las operaciones realizadas en cada uno de ellos. Esto se debe a que las distintas funciones que han sido medidas necesitan la publicación de un nuevo bloque para realizarse. Esto significa que el tiempo mostrado en la tabla se ve afectado por el intervalo de publicación de bloques (4 segundos) sumado al tiempo de aceptación de la transacción, el cual va variando en función del tráfico de la red P2P y el precio que se pague por la transacción. Esta situación provoca que el tiempo total no dependa del tiempo de ejecución del

Cuadro 7.1: Tiempos medios de ejecución de los protocolos (s).

	Simple sin ID	**Simple con ID**	**Multiparte**
Despliegue	4,83	5.47	5.33
Finalización (Cancelled)	6.13	4.2	4.7
Finalización	5.73	4.73	4.33
Cancelación	4.6	5.07	5.8
Cancelación (Finished)	4.7	5.33	5.13

código del contrato, ya que este no es tan elevado como para no encontrarse totalmente influenciado por los tiempos de aceptación y publicación.

Asimismo, un análisis interesante que se puede realizar es la comparación entre los distintos tiempos de despliegue. Esto se debe principalmente al hecho de que el despliegue del Smart Contract Simple sin ID tiene que llevarse a cabo para cada uno de los mensajes, por lo tanto, el tiempo es un factor a tener en cuenta. En cambio, el resto de contratos llevan a cabo el despliegue una sola vez durante el registro de la TTP en el sistema, y por ello, el tiempo de despliegue no es relevante en estos protocolos.

Finalmente, es importante explicar que, al haber realizado las pruebas sobre la red Kovan hay que tener en cuenta que, más allá de la diferencia de tiempo de publicación de bloques entre esta blockchain y la principal de Ethereum, la cantidad de tráfico existente en cada una de las redes también es muy distinta. En la red de Kovan, el número de transacciones es muy reducido, por lo que las nuevas son aceptadas rápidamente, y por tanto el tiempo total de ejecución se acerca al tiempo de publicación del bloque. Sin embargo, en la red principal de Ethereum, el tiempo de aceptación de una transacción puede ser mucho mayor al tiempo de publicación de un bloque ya que el número de transacciones es muy elevado. Por tanto, en un entorno real, los valores temporales serían considerablemente más elevados de los mostrados en la tabla 7.1.

7.2.2 Costes de Ejecución

Esta sección pretende determinar el coste de gas para cada una de las funciones de los distintos contratos. A diferencia del tiempo de espera, el coste de gas proporcionará una información mucho más clara y útil sobre los contratos, permitiendo comparaciones entre ellos.

Las medidas mostradas en la tabla 7.2 corresponden a los costes *fijos* de gas de cada uno de los métodos que contienen los contratos. Adicionalmente, se ha añadido a modo orientativo el precio en dólares americanos de cada una de las funcionalidades teniendo el cuenta el valor al cambio entre monedas a 23 de junio de 2018. Es importante mencionar que estos costes se han realizado teniendo en cuenta unos precios por transacción de 1 Gwei y 20 Gwei (valor máximo). La principal diferencia, además del coste total de la transacción, es el tiempo que tardará la transacción en ser aceptada por un nodo minero. De este modo, una transacción realizada sobre la blockchain principal de Ethereum con un valor de 1 Gwei puede tardar en realizarse casi 100 minutos (dependiendo del tráfico actual de la red), mientras que en el caso de 20 Gwei puede verse reducido hasta el medio minuto [22]. Por tanto, como mayor sea el precio

Cuadro 7.2: Coste de ejecución de los Smart Contracts

	Simple sin ID			Simple con ID			Multiparte		
	Gas	USD (1 Gwei)	USD (20 Gwei)	Gas	USD (1 Gwei)	USD (20 Gwei)	Gas	USD (1 Gwei)	USD (20 Gwei)
Despliegue	715.815	0.33	6.69	827.862	0.39	7.73	1.816.641	0.85	16.97
Finalización (Cancelled)	68.801	0.03	0.68	72.298	0.04	0.71	73.498	0.04	0.72
Finalización	532.266	0.26	5.24	579.563	0.29	5.70	623.351	0.31	6.13
Cancelación (Finished)	25.880	0.01	0.25	27.778	0.01	0.27	29.566	0.01	0.29
Cancelación (1 usuario)	44.698	0.02	0.44	102.011	0.05	1.00	146.772	0.07	1.44
Cancelación (10 usuarios)	-	-	-	1.020.110*	0.50	10.04	712.707	0.35	7.01
Cancelación (50 usuarios)	-	-	-	5.100.550*	2.51	50.19	3.198.072	1.57	31.47

*Cancelación realizada como N intercambios individuales.

que se pague por una misma transacción, menor será el tiempo de aceptación de la transacción.

Respecto al análisis de los resultados obtenidos, en primer lugar puede apreciarse como las funciones más costosas son las de despliegue de contratos. Estas tienen un coste de gas considerablemente superior a cualquier otra función (salvo cancelaciones de múltiples usuarios). Además, se aprecia un aumento en el coste de despliegue proporcional a la complejidad y longitud del código. Esto se debe a que una transacción para despliegue de contratos tiene un coste fijo de 31.000 unidades de *gas*, a lo que además hay que añadirle 200 unidades de gas por cada octeto que conforma el *bytecode*. Por lo tanto, como mayor longitud tenga el contrato, mayor será su *bytecode* al compilarlo, y por tanto, necesitará más gas para su despliegue. A todos estos costes también hay que sumarle el coste de ejecución de la lógica introducida en el *constructor*, aunque supone la menor parte del total en estos ejemplos.

Otro aspecto a resaltar es que un mismo método puede tener costes distintos, a pesar de ser ejecutado con los mismos parámetros. Esto se debe a que al hacer uso de estructuras *if else* la cantidad de código que se ejecuta no es siempre la misma. Por ello, se pueden dar situaciones distintas. Como puede apreciarse, la invocación del método finalización si el intercambio se encuentra cancelado no es la misma que si no lo está. Esto se debe a que en caso de encontrarse cancelado, no es necesaria la realización de ninguna acción salvo la invocación del correspondiente evento. En cambio, si no se encuentra cancelado, es necesario almacenar h_B y la clave k cifrada, lo cual supone un coste bastante alto debido a la longitud de estos parámetros. En [23] puede verse el coste de almacenamiento exacto en el Smart Contract por cada byte de información. Adicionalmente, también es notable como los contratos con identificador y multiparte tienen un coste de finalización mayor. Esto es lógico debido a que, además del almacenamiento de los parámetros, es necesaria la creación de la instancia del mensaje dentro del contrato.

Finalmente aparece el método de cancelación. Tal y como era de esperar, el coste también depende del estado en el que se encuentre la transacción. Como consecuencia, será menos costoso realizar la cancelación cuando el intercambio ya se encuentra finalizado. Otro factor destacable en el intercambio multiparte es que el coste se encuentra directamente afectado por el número de destinatarios que se estén cancelando. De esta

manera, como mayor sea el número, mayor será el coste total. Tal y como se ha podido estimar, el coste por usuario es de aproximadamente 65.000 unidades de gas.

Una vez establecidos los costes de gas por método, se procederá a la comparación entre protocolos. Como ya se ha detallado, el contrato sin identificadores es el que presenta un menor coste de gas por despliegue. No obstante, es el menos óptimo debido a que la TTP deberá desplegar un contrato para cada uno de los intercambios, incluso si no se producen disputas y el contrato no es utilizado posteriormente. Asimismo, es importante mencionar que en el caso del contrato sin identificadores, al ser necesario el despliegue del contrato para poder realizar la transacción, el tiempo de despliegue es un factor fundamental, por lo que es aconsejable un valor de transacción de 20 Gwei. Sin embargo, en el resto de contratos, el despliegue se lleva a cabo una sola vez durante el registro de la TTP y el tiempo no es un factor fundamental, por lo que podría utilizarse un valor de 1 Gwei en la transacción y producirse así un ahorro considerable. Por tanto, estos resultados, añadidos a las limitaciones del protocolo analizadas anteriormente, determinan la poca viabilidad del contrato sin identificadores.

Habiendo analizado ya las limitaciones del contrato sin identificadores, se comparará el contrato *simple con identificadores* con el *multiparte*. En primer lugar, se puede observar como el coste de despliegue en el contrato multiparte es de aproximadamente el doble. Sin embargo, es importante considerar que este desembolso se realizará una sola vez y permitirá la gestión de múltiples intercambios. Además, en el caso de realizar la transacción más lenta posible, la diferencia de precio es ínfima.

Del mismo modo, el subprotocolo de finalización tampoco supone una diferencia notable entre los protocolos. Como puede verse, los costes de gas de ambos contratos son muy similares ya que las finalizaciones, incluso en el caso del contrato multiparte, se realizan de manera individual.

El factor diferencial entre los contratos, por tanto, será el subprotocolo de cancelación. Tal y como se ha explicado a lo largo del presente documento, el contrato multiparte permite la cancelación con múltiples destinatarios de manera simultánea. Como ya se ha expuesto, el coste de cancelación por usuario es de aproximadamente 65.000 unidades de gas, a lo que se debe añadir un coste fijo de transacción. Por otra parte, en el caso del contrato simple, cada cancelación individual tiene un importe de 102.011 unidades. Esto supone una diferencia de aproximadamente 35.000 unidades por cada usuario, a excepción del primero. Por tanto, teniendo en cuenta que para este método el tiempo sí es un factor determinante y que, por tanto, es recomendable una transacción con un valor de Gwei elevado, esta diferencia puede considerarse notable, especialmente a medida que el número de receptores aumenta.

Teniendo esto en cuenta, puede concluirse que para intercambios individuales es más óptimo el uso del contrato simple con identificadores, puesto que además de presentar un menor coste de despliegue, también tiene costes de cancelación menores para un único destinatario. Por otro lado, para intercambios entre un remitente y múltiples destinatarios es recomendable el uso del contrato *multiparte*. Esto se debe a que, a pesar de tener un mayor coste de despliegue, es importante considerar que las cancelaciones las lleva a cabo únicamente el remitente y es él quien debe afrontar el pago. Por tanto, una mayor inversión inicial puede reducir a la larga el coste sobre los usuarios finales.

Finalmente, es importante tener en cuenta que es posible que los niveles de consumo de gas puedan verse reducidos de producirse un análisis más exhaustivo del código.

Asimismo, el valor en dólares debe considerarse únicamente a nivel orientativo, puesto que puede verse sujeto a fluctuaciones del valor de la moneda.

CAPÍTULO 8

CONCLUSIONES

A lo largo de las últimas décadas, con el objetivo de emular las propiedades que presenta el correo certificado ordinario, se han intentado desarrollar protocolos de correo electrónico y notificaciones certificados. Este desarrollo ha sido posible mediante el uso de funciones criptográficas que permitían emular las propiedades del correo ordinario.

Recientemente, a través del desarrollo de nuevas tecnologías como la *blockchain* se han abierto nuevas posibilidades para este tipo de intercambio, permitiendo así la obtención de nuevas propiedades.

En este documento se han presentado dos protocolos distintos de intercambio de notificaciones certificadas mediante el uso de *blockchain*. El primero, basado en el expuesto por M. Mut et al. en ”Notificaciones Certificadas sobre Blockchain"[1], expone el proceso de intercambio entre un remitente y un único destinatario garantizando las propiedades de NRR y NRO mediante el uso de una TTP. Consiguiendo mediante la *blockchain* eliminar la necesidad de que la TTP mantuviera el estado de los intercambios. El segundo protocolo detallado se ha realizado como resultado de una evolución del primero, en la que se permite el intercambio entre un remitente y múltiples destinatarios, garantizando también las propiedades de NRR y NRO.

Posteriormente, se ha mostrado el resultado principal del proyecto, correspondiente a la implementación de los *Smart Contract* que permitían el funcionamiento de los protocolos expuestos anteriormente. Por su parte, se ha finalizado la implementación mediante la integración de los contratos en una aplicación completamente funcional que permitiese la ejecución del protocolo completo.

Finalmente, se ha llevado a cabo un análisis de las propiedades proporcionadas por la aplicación y se han analizado el coste y tiempo de espera introducidos por la *blockchain* en el protocolo.

8.1 Desarrollo futuro

A continuación se detallaran los posibles vías de desarrollo que han quedado abiertas, y que supondrían una mejora en los protocolos propuestos:

- Realizar pruebas de ejecución sobre la red *Ethereum* para poder medir en detalle los tiempos de espera y analizar los posibles efectos negativos de estas.
- Optimizar el código con la intención de reducir al mínimo los costes de gas innecesarios.
- Modificar la implementación del protocolo de intercambio para poder obtener las propiedades de *equidad fuerte* y *transferibilidad de las pruebas*.
- Adecuar el protocolo para impedir las confabulaciones de usuarios con la TTP que den lugar a comportamientos no deseados.
- Integrar el Smart Contract sobre una aplicación de escritorio ad hoc en lugar de la aplicación web que permita prescindir de la base de datos centralizada para los intercambios.
- Implementar el uso de listas negras para sufragar ataques de denegación de servicio. Del mismo modo, realizar un análisis sobre los costes de gas derivados de su implementación y analizar su rentabilidad en este tipos de ataque.

BIBLIOGRAFÍA

[1] M. Mut-Puigserver, M. M. Payeras-Capellà, and M. A. Cabot-Nadal, "Notificaciones certificadas sobre blockchain," *Actas de la XV Reunión Española de Criptología y Seguridad de la Información*. (document), 1, 2.1.3, 2.4, 4, 7.1.1, 7.1.1, 7.1.3, 8

[2] M. M. Payeras Capellà, "Protocols de comerç electrònic: Pagament anònim i intercanvi equitatiu," *Universitat de les Illes Balears*, pp. 103–123, 2005. 1.1, 2.1, 2.1.1, 2.1.2, 2.1.3, 5, 5.5, 7.1.3

[3] "Smart contracts," *Investopedia*. [Online]. Available: https://www.investopedia.com/terms/s/smart-contracts.asp 1.2

[4] J. Ferrer-Gomila, J. Onieva, M. Payeras-Capellà, and J. Lopez, "Certified electronic mail: Properties revisited," *Computers and Security, vol 29n nº2*, 2010. 1.2, 2.1.3, 2.2, 7.1.3

[5] S. Kremer, O. Markowitch, and J. Zhou, "An intensive survey of fair non-repudiation protocols," *Computer Communications*, 2002. 2.2

[6] O. Markowitch and Y. Roggeman, "Probabilistic non-repudiation without trusted third party," *Second Workshop on Security in Communication Network*, 1999. 2.2

[7] C. Petrucci, F. Gennai, A. Shahin, and A. Vinciarelli, "La posta elettronica certificata," *RFC 6109*, 2011. 2.2

[8] J. L. Ferrer-Gomila, M. Payeras-Capellà, and L. H. i Rotger, "An efficient protocol for certified electronic mail," *Departament de Ciències Matemàtiques i Informàtica, Universitat de les Illes Balears*. 2.2, 2.4, 4, 4.4, 4.5.2, 4.5.2, 4.6, 7.1.1, 7.1.2

[9] J. A. Onieva, J. Zhou, and J. Lopez, "Multi-party non-repudiation: A survey," *ACM. Comput Surveys, vol 41 NCIS Lab. Publications: https://www.nics.uma.es/publications*, 2008. 2.3, 2.3

[10] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Towards fariness of cryptocurrency transactions," *Aalto University, Finland and NEC Laboratories, Germany*. 2.4

[11] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *www.bitcoin.org*, pp. 1–9, 2008. 3

[12] M. Crosby, Nachiappan, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: Beyond bitcoin," *Applied Innovation Review*, pp. 7–12, 2016. 3, 3.1

[13] J. Siim, "Proof-of-stake," *Research Seminar in Cryptography*. [Online]. Available: https://courses.cs.ut.ee/MTAT.07.022/2017_fall/uploads/Main/janno-report-f17.pdf 3.1

[14] C. oficial Ingenieros de Telecomunicaciones, "Ethereum. infraestructura descentralizada para smart contracts," *Curso on-line en criptomoneda y Smart Contracts*, 2017. 3.2, 3.2.1, 3.2.2

[15] "Solidity." [Online]. Available: https://solidity.readthedocs.io 6.1.1

[16] "Remix, ethereum-ide." [Online]. Available: https://remix.readthedocs.io 6.1.1

[17] "Metamask, how it works." [Online]. Available: https://metamask.io/ 6.1.1

[18] "Stanford javascript crypto library." [Online]. Available: http://bitwiseshiftleft.github.io/sjcl/doc/ 6.1.2

[19] "Developing ethereum smart contracts for begginners," *Coursetro*. [Online]. Available: https://coursetro.com/courses/20/Developing-Ethereum-Smart-Contracts-for-Beginners 6.7

[20] "Kovan testnet." [Online]. Available: https://kovan-testnet.github.io/website/ 7.2.1

[21] "Etherscan. kovan testnet." [Online]. Available: https://kovan.etherscan.io/ 7.2.1

[22] "Gas station." [Online]. Available: https://ethgasstation.info/ 7.2.2

[23] L. Senta, "Where and how application data is stored in ethereum?" 2017. [Online]. Available: https://blog.singulargarden.com/posts/storage-and-dapps-on-ethereum-blockchain/ 7.2.2