Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones

Trabajo Fin de Máster

Estudio y despliegue de un esquema de cheque bancario basado en Corda

Autor: Óscar de Heras Horcajada

Directora: María Francisca Hinarejos Campos

Índice

Índice 3
Índice de figuras 5
Resumen 7
1. Introducción 9
2. ¿Qué es Blockchain? 11
2.1 Corda 15
2.1.1 Redes Corda 16
2.1.2 La ledger en Corda 17
2.1.3 Transacciones Corda 18
2.1.4 Smart contracts en Corda 21
2.1.5 Consenso y notarios en Corda 22
2.1.6 Aplicaciones Corda (CorDapp) 24
3. Análisis del sistema 25
3.1 Intervinientes, metodologías y objetos 25
3.2 Requerimientos de seguridad 27
3.3 ZKP (Zero Knowledge Proof) 29
3.4 Consideraciones generales 31
3.5 Topología del sistema 40
3.6 Protocolo de emisión de un cheque 43
3.6.1 Pasos del protocolo 43
3.6.2 Incidencias o motivos de rechazo en la emisión 44
3.7 Protocolo de transferencia de un cheque 46
3.7.1 Pasos del protocolo 46
3.7.2 Incidencias o motivos de rechazo en la transferencia 48
3.8 Protocolo de depósito de un cheque 50
3.8.1 Pasos del protocolo 50
3.8.2 Incidencias o motivos de rechazo en el depósito 51
4. Prueba de concepto 54
4.1 Entorno de pruebas 55

4.2 Estructura y análisis de la prueba de concepto 56
4.2.1 TFM-Poc_CordaNode 58
4.2.2 TFM_PoC_RPCClients 65
4.3 Resultados tests PoC 66
5. Planificación 77
5.1 Planificación inicial 78
5.2 Planificación final 80
6. Conclusiones 82
7. Futuras mejoras 84

Índice de figuras

Figura 1. Cadena de bloques 12
Figura 2. Ejemplo topología red Bitcoin 12
Figura 3. Ejemplo transacción Bitcoin 13
Figura 4. Ejemplo Smart contract en Java 15
Figura 5. Red Corda 17
Figura 6. Ledgers en Corda 18
Figura 7. Estados de entrada y salida en una transacción Corda 19
Figura 8. Clasificación de transacciones Corda 20
Figura 9. Ejecución de contratos 21
Figura 10. Notario vs. Clúster de notarios 22
Figura 11. Clústeres de notarios 23
Figura 12. Flujo CorDapp 24
Figura 13. IZKP (Prueba de conocimiento nula interactiva) 30
Figura 14. NIZKP (Prueba de conocimiento nula no interactiva) 31
Figura 15. Topología del sistema 41
Figura 16. Protocolo de emisión 45
Figura 17. Protocolo de transferencia 49
Figura 18. Protocolo de depósito 53
Figura 19. Estructura proyecto TFM-PoC_CordaNode 58
Figura 20. Configuración nodo Corda en la tarea deployNodes de build.gradle 61
Figura 21. Configuración de los nodos Corda Notary y Bank2 usados en la PoC 62
Figura 22. Script recreate_full.sh 63
Figura 23. Estructura NetworkParameters 64
Figura 24. Ejecución de nodo Corda en JVM 64
Figura 25. Escala de tiempo. Planificación inicial 79
Figura 26. Diagrama de Gantt. Planificación inicial 79
Figura 27. Escala de tiempo. Planificación final 81
Figura 28. Diagrama de Gantt. Planificación final 81

Resumen

Este proyecto desarrolla un sistema que permite la emisión, transferencia y depósito de cheques al portador de forma segura. El sistema será capaz de asegurar la equitatividad en cada uno de los procesos, así como, el anonimato de los usuarios que participan en el proceso de transferencia.
Además, se detectará todo intento de transferencia o depósito de un cheque previamente transferido o depositado.

Para cumplir con lo anterior se utilizará Corda [9]-[12]. Corda es una plataforma Blockchain desarrollada por R3 que permite crear redes privadas donde la adhesión a las mismas debe ser autorizada por un organismo que regula la operativa del sistema.

Así mismo, el sistema hace uso de una tercera parte de confianza, la cual no pertenece a la red Corda, que durante el proceso de transferencia sirve a los usuarios como organismo al que solicitar dicha transferencia.

1. Introducción

El objetivo de este proyecto es proporcionar un sistema, basado en alguna de las plataformas Blockchain [2] existentes, que permita la emisión, transferencia y depósito de cheques al portador.

Aunque desde la aparición de Blockchain muchas han sido las plataformas que han surgido ninguna de ellas gozaba de buena reputación en el sector financiero, esto ha provocado un rechazo y una escasa implantación de sistemas Blockchain en las instituciones de dicho sector. Corda, una de las últimas plataformas en surgir, nació con un claro objetivo: Ofrecer un sistema basado en Blockchain, que resuelva muchas de las carencias que hasta hoy tenían los anteriores, y que fuera capaz de ofrecer ciertas garantías y características que lo hicieran interesante a instituciones financieras.

El esquema que este documento presenta usará Corda como medio para asegurar su correcto funcionamiento, permitiendo crear una red privada a la que se adherirán los bancos que quieran ofrecer a sus clientes la emisión y el depósito de cheques bancarios al portador. Además, permitirá la transferencia de un cheque previamente emitido tantas veces como se desee sin que la identidad de los usuarios que intervienen en dicha transferencia llegue a ser conocida por ningún banco del sistema.

El esquema, en cada uno de sus protocolos, definirá medidas que aseguren la equitatividad de las partes y, mecanismos que le protejan ante un posible fallo o intento de actuación fraudulenta de algún sujeto presente, o no, durante la emisión, transferencia y depósito. Esto implica que el sistema tendrá que ser capaz de detectar y evitar:

- El intento de emisión de un cheque por un banco haciéndose pasar por otro.
- Que el tenedor de un cheque intente su transferencia más de una vez o que intente su depósito una vez lo ha transferido.
- Que el tenedor de un cheque intenté su depósito más de una vez.

En resumen, es objetivo de este documento definir los protocolos que van a regir la emisión, transferencia y depósito y que van a permitir un funcionamiento normal del sistema. También es objetivo que por cada protocolo se estudien los posibles casos de fallo que impidan su finalización, así como la resolución de las incidencias que puedan surgir.

El objeto de cada uno de los capítulos de esta memoria es:

- **Capítulo 2:** Introducción a Blockchain y a Corda.
- **Capítulo 3:** Estudio y análisis teórico del esquema.
- **Capítulo 4:** Prueba de concepto del sistema desarrollado.
- **Capítulo 5:** Planificación inicial y final del proyecto.
- **Capítulo 6:** Conclusiones de este Trabajo Final de Máster.

- **Capítulo 7:** Futuros desarrollos o mejoras del sistema.

2. ¿Qué es Blockchain?

Aunque hoy en día es un término bastante popularizado, y del que cada vez oímos hablar más, su origen es bastante reciente. Fue en el año 2008 cuando Satoshi Nakamoto publicó: "*Bitcoin: A Peer-to-Peer Electronic Cash System*" [2]. En este documento se presentaba la criptomoneda *Bitcoin* [1] y la tecnología que, sin duda, iba a revolucionar y a soportar nuevos sistemas transaccionales descentralizados y de total confianza: *Blockchain* [7]- [8].

Antes de esta publicación, en el año 1990, Stuart Haber y W. Scott Stornetta publicaron un artículo, en la revista *Journal of Cryptology*, llamado "*How to time-stamp a digital document*" [6], que planteaba la necesidad de un modelo que permitiese el almacenaje de documentos, para que fueran manejados por más de una persona, y el uso del fechado digital para así poder detectar cualquier modificación en un documento. Fue este entonces el primer artículo en el que se hablaba sobre la necesidad de un sistema similar al que presentó en el año 2008 Satoshi Nakamoto.

Blockchain - cadena de bloques - se presenta como una tecnología capaz de almacenar todas las transacciones llevadas a cabo en un sistema que hace uso de dicha tecnología. Además, permite que mediante mecanismos de consenso se puedan ratificar todas las transacciones y detectar cualquier modificación futura que se hiciese sobre una transacción ratificada anteriormente.

Es en los mecanismos de consenso donde reside el verdadero potencial de Blockchain. Estos permiten que la validación y el almacenamiento de las transacciones no recaiga en una única entidad si no en un conjunto de entidades. ¿Qué aporta el tener un sistema de validación descentralizado y almacenamiento distribuido a una plataforma basada en Blockchain?

- **Mayor disponibilidad del servicio**. Siempre es más difícil atacar a un conjunto de equipos que a uno solo.
- **Mayor integridad de datos**. Un almacenamiento distribuido hace más difícil el alterar datos de transacciones ya efectuadas y, en caso de producirse, permite detectar y subsanar la alteración.
- **Mayor confianza**. Que la validación de las transacciones no recaiga en una sola entidad permite aumentar el nivel de confianza en el sistema frente a posibles actuaciones fraudulentas.

La forma que tiene Blockchain de almacenar las transacciones es en bloques que a su vez son almacenados de forma secuencial y cronológica en la cadena. La forma de asegurar el orden de los bloques es almacenando en cada uno de ellos una referencia a su boque anterior.

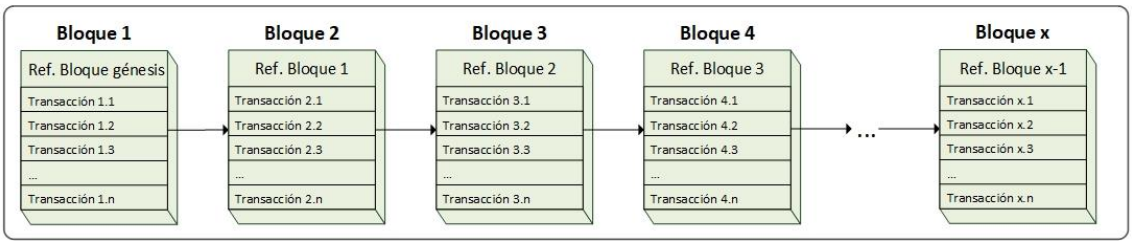


Figura 1. Cadena de bloques

En un entorno Blockchain la topología de la red y la disposición de sus nodos y conexiones variará según la implementación de la plataforma. Por ejemplo, en Bitcoin, todos los nodos conectados a la red son iguales y solo es posible de hacer una clasificación de ellos en base las funcionalidades que soportan. Estas funcionalidades, y su cometido, son:

- **Wallet**: Actúa como monedero de Bitcoins (*BTC*). Tienen asociada una dirección única (por ejemplo: 3FZbgi29cpjq2GjdwV8eyHuJJnkLtktZc5) necesaria para la transferencia de *BTC*.
- **Minero**: Participa en las pruebas de trabajo (*PoF – Proof of Work*) [5] y compite con otros mineros para crear bloques y obtener la recompensa (*BTC*) correspondiente. Posibilita que el sistema sea descentralizado.
- **Almacenamiento de transacciones:** Guardan una copia completa de la Blockchain. Posibilita que el sistema sea distribuido.
- **Enrutamiento de red**: Se encarga de encaminar y distribuir las transacciones hacia otros nodos de la red. Es importante su eficiencia para asegurar una correcta distribución de las transacciones.

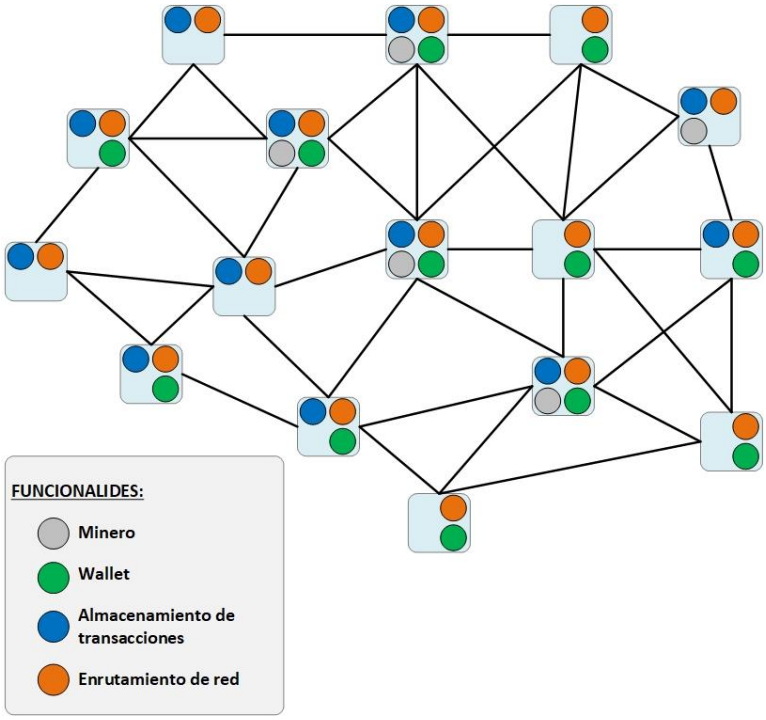


Figura 2. Ejemplo topología red Bitcoin

Blockchain permite el despliegue de redes privadas donde el acceso a las mismas es controlado y, por tanto, se ofrece una mayor seguridad a los miembros de dicha red. Por el contrario, este tipo de redes, puede hacer que se pierda cierto grado anonimato y descentralización si para su acceso se requiere de un registro. Blockchain también permite el despliegue de redes públicas. En estas la conexión a la red no es controlada y cualquiera puede acceder y operar en ella simplemente descargando y ejecutando el software necesario.

Respecto a cómo se opera con Blockchain dependerá del objetivo para el cual fue creada la plataforma y, por tanto, de la implementación que se hizo. Siguiendo con Bitcoin, la figura 3 pretende mostrar cómo sería una transferencia de *BTC* entre dos usuarios.

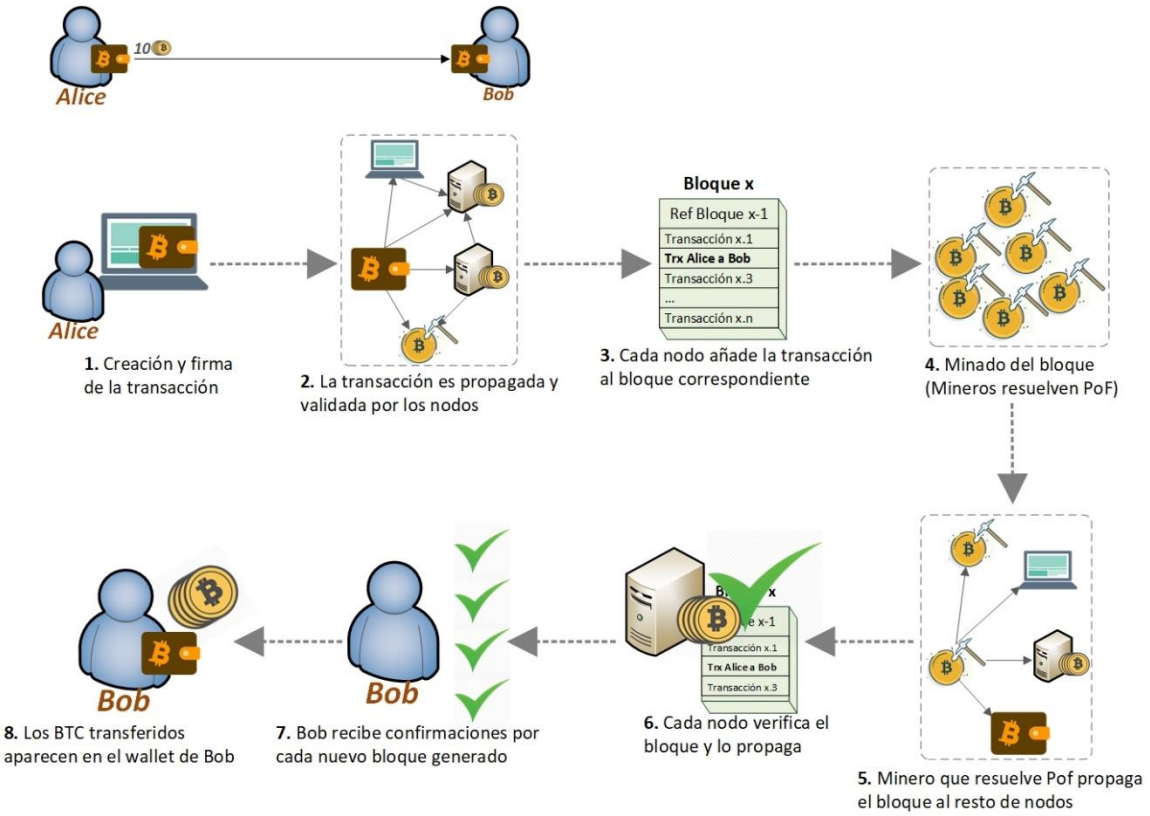


Figura 3. Ejemplo transacción Bitcoin

Aunque dependiendo de la plataforma el uso de la tecnología Blockchain puede variar, sí que podemos identificar algunas características comunes a la mayoría de plataformas. Estas características son:

- Se trata de un sistema descentralizado donde cada nodo se ejecuta de forma autónoma.
- La cadena puede ser compartida y distribuida entre todos, o parte de, los miembros de la red originando un sistema descentralizado (*DLT – Distributed Ledger Technology*).

- Se permite ejecutar transacciones *peer-to-peer*, ofreciendo anonimato a las partes involucradas, sin necesidad de que un tercero de confianza (*Trusted Third Party – TTP*) conocido supervise ni valide la operación.
- Los bloques deben ser confirmados antes de darse por buena su concatenación a la cadena.
- Una vez el bloque, y por tanto sus transacciones, han sido confirmadas no podrán ser modificadas en el futuro sin que este hecho sea detectado.
- En cualquier momento la cadena puede ser auditada, comprobando que los bloques y las transacciones que contiene son válidas.
- Tecnología flexible. Permite almacenar en los bloques cualquier tipo de información con lo que se aumentan los casos de uso de la misma.
- Uso de criptografía simétrica, asimétrica y de algoritmos de hash.

Hasta ahora muchas de las transacciones clásicas de, por ejemplo, carácter financiero estaban reguladas por un contrato que debía ser aceptado por todas las partes involucradas en dicha transacción y, que definía lo que había que hacer y como había que hacerlo. En muchos casos estos contratos eran costosos documentos legales que requerían de un coste económico y de tiempo para su elaboración, en otros casos esos contratos se creaban partiendo de unas plantillas definidas lo que hacía ahorrar costes y tiempo.

En septiembre de 1997 *Nick Szabo* publicó el artículo "*Formalizing and Securing Relationships on Public Networks*" [3], este recogía la necesidad de digitalizar los contratos legales tradicionales para crear protocolos que, basados en criptografía y sin necesidad de una tercera parte de confianza o de un organismo centralizado que validase la relación, permitiesen a extraños participar en una misma transacción a través de internet. Debido a la tecnología del momento no se pudo implementar ninguna prueba de concepto que, de forma práctica, ofreciese unos resultados que avalasen lo publicado por *Szabo*.

Con la aparición de Blockchain en 2008 se vio el entorno ideal, sistema descentralizado con datos distribuidos, para empezar a pensar en la posibilidad de transformar los contratos tradicionales en contratos inteligentes siguiendo las premisas que escribió *Szabo* en su artículo. Desde entonces muchas de las plataformas, basadas en Blockchain, que han surgido han estado orientadas hacia un modelo que haga un uso intensivo y flexible de los *smart contracts*.

Un *smart contract* no es más que un programa informático, ver figura 4, donde los términos del contrato se corresponden con sentencias del código que lo forma, que es ejecutado de forma autónoma y automática durante una transacción y que tiene como objetivo hacer cumplir lo acordado entre las partes y/o lo definido por el sistema. Dicho *smart contract* puede no residir en ninguna de las partes que intervienen en la transacción, residir solo en algunas o residir en todas, e incluso, puede que su código

fuente sea conocido. El hecho de no estar centralizado permite que una transacción no se vea comprometida por el simple hecho de modificar el código de un *smart contract*.

```
final Check checkInput = tx.inputsOfType(Check.class).get(0);
final Check checkOutput = tx.outputsOfType(Check.class).get(0);
final Party issuerBank = checkInput.getIssuerBank();

// Constraints on the shape of the transaction
if (tx.getInputs().isEmpty())
    throw new IllegalArgumentException("Error: One transaction must be consumed when transfering a check\n\n");
if (!(tx.getOutputs().size() == 1))
    throw new IllegalArgumentException("Error: One check must be generated\n\n");

// Constraints on the check data
if (checkInput.getExpDate().isBefore(new Date().toInstant()))
    throw new IllegalArgumentException("Error: The check is already expired ("+checkInput.getExpDate()+
            ") so it can not be transfered\n\n");
if(checkOutput.getOpId() == null)
    throw new IllegalArgumentException("Error: The output check has to define an Id Operation\n\n");
if(checkInput.getOpId().equals(checkOutput.getOpId()))
    throw new IllegalArgumentException("Error: The input check and the output check can not have the same Id Operation "
            + "("+checkInput.getOpId()+")\n\n");

// Constraints on the signers
final List<PublicKey> requiredSigners = command.getSigners();
final List<PublicKey> expectedSigners = Arrays.asList(issuerBank.getOwningKey());
if (!(requiredSigners.containsAll(expectedSigners)))
    throw new IllegalArgumentException("\n\n"+issuerBank.getName()+" Error: The bank issuer must be signer.\n\n");
```

Figura 4. Ejemplo Smart contract en Java

El documento "*Smart Contract Templates: foundations, design landscape and research directions*" [4], publicado el 4 de agosto de 2016 por *Clack*, *Bakshi* y *Braine*, ofrece una definición sobre *smart contract* que abarca tanto parte la operacional, derechos y obligaciones al ejecutar un *smart contract*, como la forma en que los contratos legales pueden ser codificados en base a la interpretación que se haga de ellos. Por otro lado, determina la importancia del uso de plantillas estandarizadas de *smart contracts* que sean capaz de soportar contratos complejos.

Hoy en día, aunque para la gran mayoría de usuarios, tanto Bitcoin como Ethereum son los principales referentes en cuanto a uso de Blockchain, también existen, y van surgiendo, nuevos sistemas, tantos públicos como privados, cuya funcionalidad se basa en Blockchain. Algunos de los últimos sistemas en aparecer, con una orientación claramente empresarial han sido: Hyperledger (diciembre 2015) y Corda (agosto de 2016).

2.1 Corda

Corda es una plataforma open-source de Blockchain presentada en 2016. Está desarrollada por el consorcio R3 y auspiciada por más de 200 compañías entre las que se encuentran un gran número de entidades financieras referentes a nivel mundial: Barclays, BBVA, Bank of America, ING, Bank of Canada...

Dado el perfil de muchas de las compañías que colaboran con el desarrollo de Corda, uno de los objetivos que persigue el proyecto es hacer de ésta la primera infraestructura Blockchain útil a nivel financiero, de tal forma que sea capaz de soportar muchos de los productos y servicios que ofrece la banca a día de hoy.

2.1.1 Redes Corda

Una de las características principales de Corda, que hace interesante su implantación a entidades financieras, es la posibilidad de crear redes privadas (*CBN – Corda Business Network*) donde el acceso a la misma es controlado y donde todos los miembros adheridos a dicha red se conocen entre sí.

La adhesión controlada a la red supone que los usuarios que soliciten acceso se comprometen a cumplir unas políticas y protocolos, tanto funcionales como técnicos, que rigen el funcionamiento de la red y de los servicios que en ella se ofrecen. Estas políticas son definidas por el gobernador de la red (*BNG - Business Network Governor*) y su cumplimiento es controlado por el operador de red (*BNO – Business Network Operator*). Existe entonces una relación directa entre gobernador y operador de red, encargándose este último del 'día a día' del sistema, interactuando con los usuarios de la misma y asegurando un correcto funcionamiento del sistema en general.

Las funciones que puede llevar a cabo el gobernador de la red son:

- Definir las bases legales y regulatorias sobre las que operará la red.
- Establecer el modelo financiero que sustentará el crecimiento de la red.
- Determinar cómo la red será operada y soportada.
- Determinar reglas y procesos para la gestión y resolución de disputas.

En cuanto a las funciones que puede desempeñar el operador de red están:

- Hacer que se cumplan las reglas definidas por el gobernador.
- Controlar la adhesión de usuarios a la red.
- Revocar la adhesión de un usuario de la red.
- Generar certificado X.509 para cada usuario que les identifique de manera única.
- Gestionar internamente la información de todos los usuarios de la red.
- Distribución de las aplicaciones Corda.
- Proporcionar información a los nodos sobre los miembros de la *CBN*.
- No participar en la ejecución de transacciones.

Una vez aceptado el acceso a la red el usuario pasará a ser un nodo de la red Corda (*BNU – Business Network User*). Ser un nodo implica la instalación de un software específico en un servidor y su presentación ante el operador de red para su inclusión en el sistema a efectos funcionales, esta inclusión se traduce en una adhesión en el registro de nodos y en la emisión de certificados X.509. Cada nodo de la red es identificado de manera única mediante su certificado X.509.

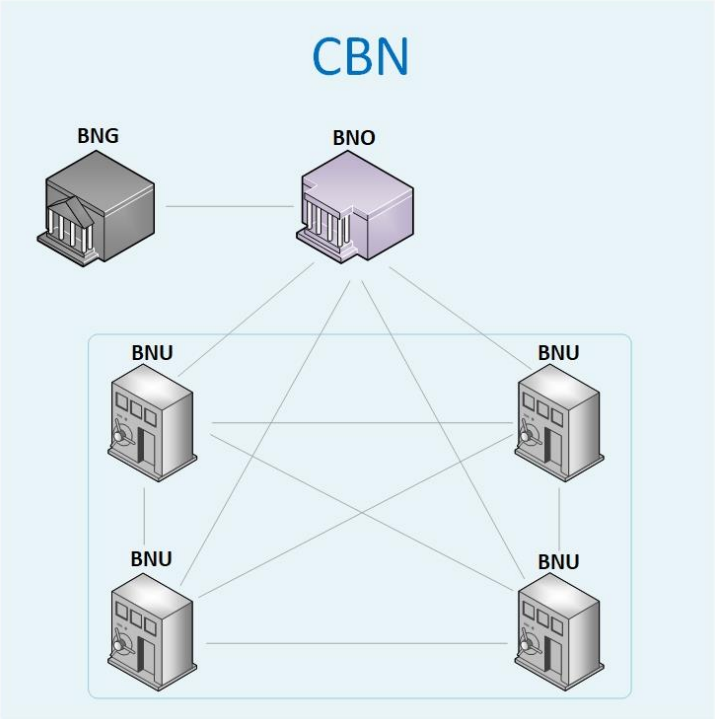


Figura 5. Red Corda

La definición de políticas en la red, y su obligado cumplimiento, ofrecen una serie de garantías legales y de conducta que hace que la confianza con el sistema y entre los nodos de la red aumente. Así mismo, la definición de protocolos de comunicación y mensajería facilita la interoperabilidad entre nodos evitando problemas de compatibilidad.

2.1.2 La ledger en Corda

Corda, en el documento "*Corda: An Introduction*" [10] publicado por *Richard Gendal Brown*, *James Carlyle*, *Ian Grigg* y *Mike Hearn* en agosto de 2016 se presenta como una plataforma de contabilidad distribuida para almacenar y procesar acuerdos financieros, que soporta el uso de *smart contract*. Esta definición contiene otras dos de las principales características de Corda: existencia de un libro de cuentas distribuido, ledger de aquí en adelante, que almacena las transacciones en las que participa el usuario y posibilidad de implementar y ejecutar *smart contracts* que regulen las transacciones entre nodos de la red.

Lo que diferencia a Corda de otras plataformas que implantan tecnologías de contabilidad distribuida (*DLT - Distributed Ledger Technology*) es que cada nodo tendrá su propia ledger y en ella se almacenarán las transacciones en las que participa dicho nodo. A continuación se ejemplifica esto:

Supongamos una red formada por 4 nodos (Banco1, Banco2, Banco3 y Banco4) que pueden operar entre sí y donde cada uno de ellos tiene su propia ledger. Tal y como implementa Corda su tecnología de contabilidad distribuida la ledger de cada uno solo

almacenará las transacciones en las que él participa. Esto implica que, por ejemplo, en una transacción entre el Banco1 y el Banco2, esta será desconocida para el resto de bancos de la red. Cualquier usuario presente en una transacción puede afirmar que: "*I know that what I see is what you see*".

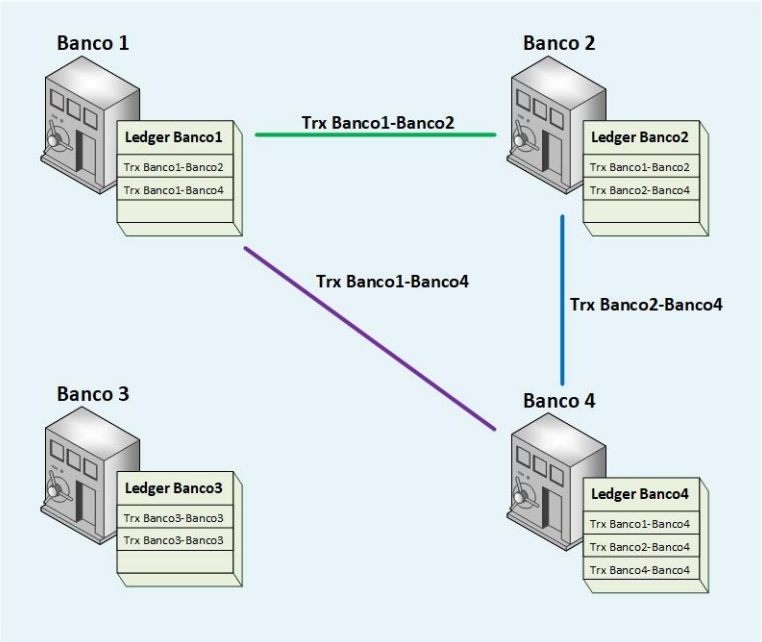


Figura 6. Ledgers en Corda

2.1.3 Transacciones Corda

Para comprender que es una transacción Corda y cuál es su funcionamiento conviene identificar un elemento intrínseco a ella: objetos estado, o estado. Un estado es un objeto inmutable creado en un instante temporal determinado que puede contener cualquier tipo de dato y que es generado, y también puede ser usado, por una transacción. En concreto un estado contiene:

- **Referencia al *smart contract*** que define la función de verificación. Cada estado debe hacer referencia a uno, o más, *smart contracts*.
- **Propiedades** que reflejan los datos y valores del estado en un instante determinado. Estos no pueden ser modificados.
- Una lista de los nodos **Participantes** en la transacción que pueden consumir este estado.

Al no poder ser modificado el contenido de un estado su validez, o no, para ser usado en una transacción viene determinada por una condición que indica si está disponible para ser consumido o no. Un estado tendrá condición de consumido cuando ya ha sido usado como entrada en una transacción que finalizó de forma satisfactoria y, no consumido cuando todavía ninguna transacción ha hecho uso de él.

Corda usa un modelo *UTXO* (*unspent transaction output*). Esto significa que una transacción puede tomar solo como datos de entrada objetos estado no consumidos, es decir, estados que no han sido usados anteriormente en otras transacciones. Así mismo la transacción puede generar nuevos estados que se marcarán como no consumidos. El hecho de que un estado sea consumido no significa que desaparezca de la ledger si no simplemente que su uso por otra transacción deja de ser posible.

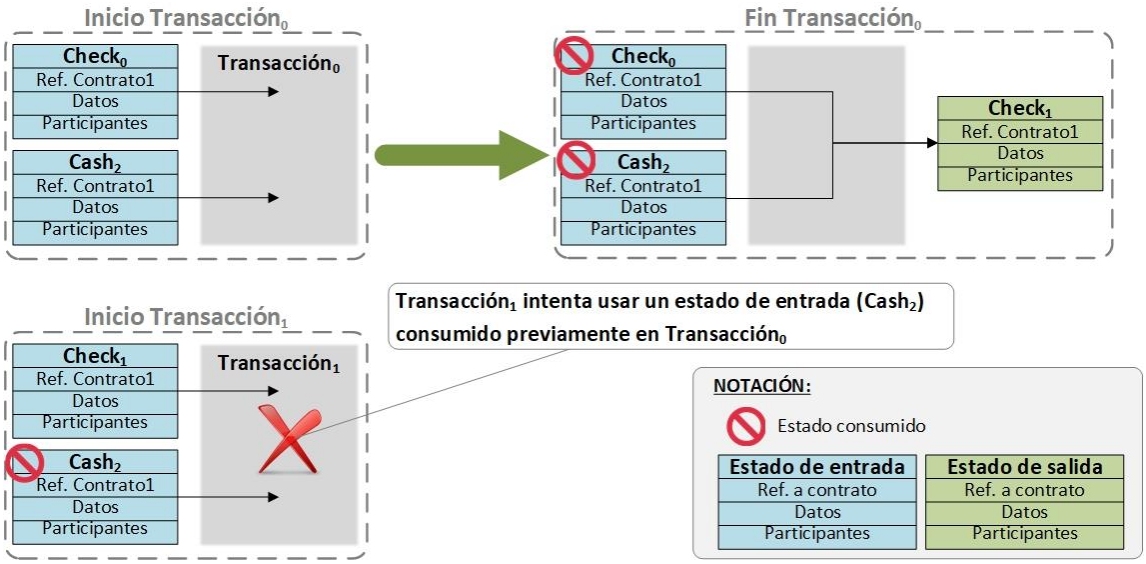


Figura 7. Estados de entrada y salida en una transacción Corda

Cada ejecución satisfactoria de una transacción supone un cambio en la ledger, según la naturaleza del cambio podemos clasificar las transacciones en:

- **Emisiones**: No consumen ningún estado de entrada, pero sí generan estados de salida.
- **Actualizaciones**: Consumen estados de entrada no consumidos anteriormente y generan estados de salida.
- **Salidas**: Consumen estados de entrada no consumidos anteriormente y no generan ningún objeto de salida.

Al iniciar una transacción se pueden definir los nodos de los que se requiere su firma para dar por terminada una transacción de tal forma que, o todos firman la transacción o esta no termina de completarse. Esto supone que un nodo hará efectiva en su ledger una transacción cuando:

- Haya recibido del resto de nodos la transacción junto con su firma correspondiente.
- El *smart contract* haya verificado de forma satisfactoria los estados de entrada y de salida de la transacción.
- Ningún estado de entrada de la transacción, en caso de definirse, haya sido consumido por otra transacción.

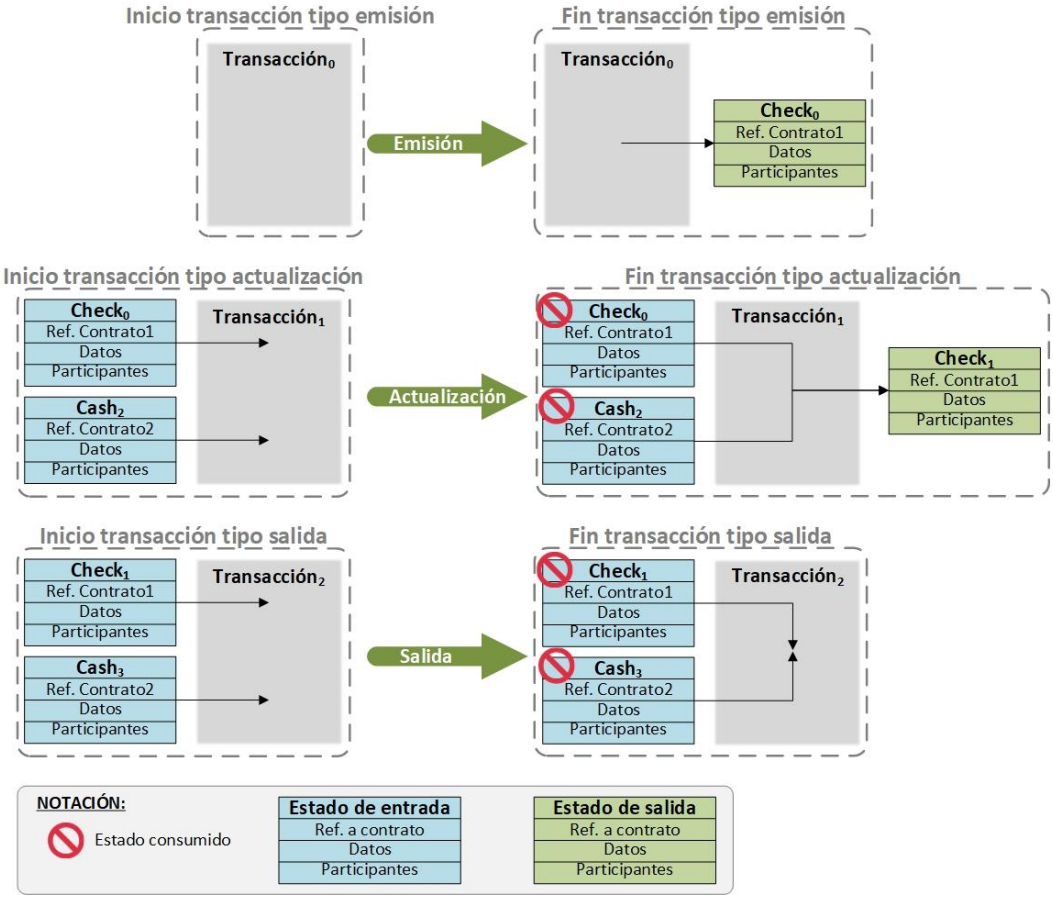


Figura 8. Clasificación de transacciones Corda

Una transacción puede incluir referencias a estados para que estos puedan ser usados en la ejecución del *smart contract*. Para su uso se comprobará que estos estados no han sido consumidos y, una vez finalizada la transacción, su condición permanece como no consumido.

En Corda todas las transacciones son atómicas, es decir, o todos los cambios que propone la transacción sobre la ledger son aceptados o ninguno de ellos lo es. Además, la atomicidad se aplica sobre todos los nodos que participan en la transacción evitándose así que se produzcan inconsistencias de estados entre nodos.

Además de los estados de entrada y de salida una transacción puede contener:

- **Comandos**: Permiten definir el propósito de la transacción y según este aplicar unas validaciones u otras.
- **Archivos adjuntos**: En ocasiones, y dependiendo de la naturaleza de la transacción, se hace necesario el adjuntar archivos que pueden ser usados al verificar la transacción.
- **Ventana de tiempo**: Permite definir un periodo de tiempo durante el cual la transacción tiene que llevarse a cabo y hacerse efectiva en la ledger.

- **Notarios**: Servicio de la red que durante la ejecución de una transacción asegura no haber firmado con anterioridad ninguna otra transacción que haya hecho uso de algunos de los estados que pretende consumir la actual.

2.1.4 Smart contracts en Corda

En Corda un *smart contract*, o contrato de aquí en adelante, es un programa escrito en lenguaje Java o Kotlin, que es capaz de ser interpretado por una máquina virtual de Java (JVM – Java Virtual Machine) y que define de forma programática un conjunto de restricciones, en base a las reglas de un contrato o acuerdo pre-acordado anteriormente.

El contracto define funciones que llevan a cabo verificaciones, en base a unas restricciones impuestas, sobre la transacción a examen. Un contrato puede validar:

- Los estados de entrada y de salida de la transacción.
- Los archivos adjuntos a la transacción.
- La ventana de tiempo en la que se está ejecutando la transacción.
- El propósito de la transacción.

Si alguna regla de las codificadas en el contrato no es satisfecha la transacción no será aceptada lo que implica que la ledger no sufrirá cambio alguno.

La ejecución de un contrato es determinista ya que, independientemente del instante de tiempo en el que se esté ejecutando y dado que los datos a verificar no pueden variar, el resultado será siempre el mismo. Como norma general el resultado de la ejecución de un contrato nunca dependerá de datos del exterior. El hecho de ser determinista permite que todos los nodos de la red puedan llegar a un consenso sobre el estado de las transacciones de sus ledgers y, en un futuro, llevar a cabo comprobaciones sobre la ledger que verifiquen que ningún estado o transacción ha sido alterado.

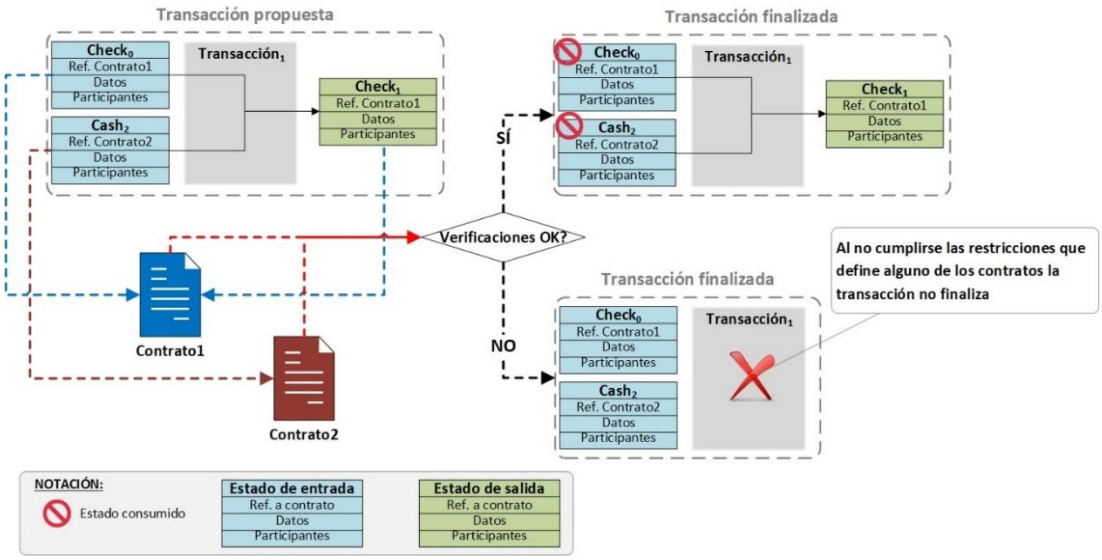


Figura 9. Ejecución de contratos

2.1.5 Consenso y notarios en Corda

Una transacción se hará efectiva en la ledger cuando se haya llegado a un consenso en cuanto a su validez y unicidad.

El consenso en la validez es alcanzado por todos los nodos cuya firma es requerida en la transacción. Este consenso se alcanza cuando:

- Todos los nodos han validado la transacción actual satisfactoriamente.
- Todos los nodos han validado la cadena completa. Esto significa que todas las transacciones previas que dieron lugar a los estados de entrada de la transacción actual deben ser validadas satisfactoriamente.
- En la transacción se han adjuntado todas las firmas requeridas.

En cuanto al consenso de unicidad, este impide que una transacción consuma un estado previamente consumido; evita, por tanto, el doble gasto de estados. El consenso de unicidad es alcanzado por el notario, o clúster de notarios, adheridos a la red.

Podemos considerar al notario, o al clúster de notarios, como nodos de confianza dentro de la red ya que participarán en todas las transacciones que necesiten consumir estados previos. Se encargan de certificar que una transacción no ha hecho uso de un estado consumido previamente, evitando así el doble uso de estados y permitiendo alcanzar un consenso de unicidad. La detección de un intento de doble uso conlleva que los notarios rechacen la transacción e informen en su respuesta el motivo del rechazo.

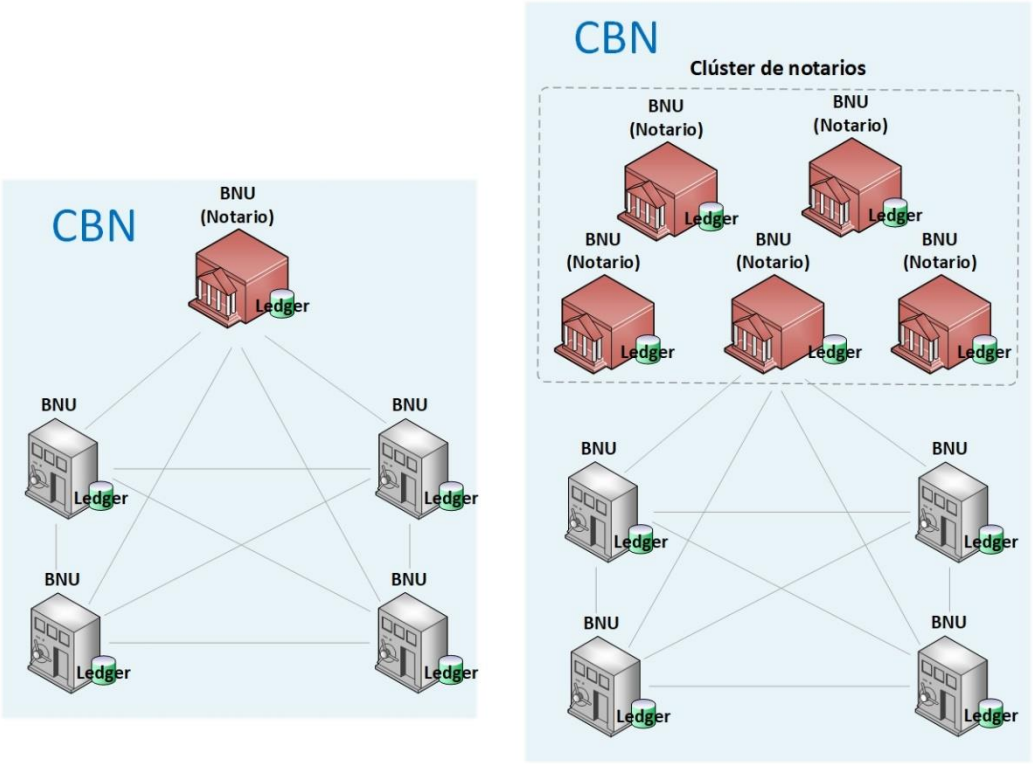


Figura 10. Notario vs. Clúster de notarios

Cuando una transacción ha definido una ventana de tiempo para su ejecución los notarios solo certificarán la transacción durante esa ventana. Opcionalmente, a los notarios también se les puede asignar la tarea de validar la transacción, en este caso actuarán como otro nodo más de la red rechazándola si fuera necesario. Permitir a los notarios validar la transacción supone una pérdida de privacidad al ser los datos de la transacción conocidos por los notarios, por el contrario, evita que un nodo pueda llevar a cabo un ataque de denegación de estados. Este ataque consiste en crear transacciones inválidas que hagan uso de estados no consumidos y que el notario las valide para dejar esos estados como consumidos.

Es en el momento de crear la transacción cuando se define el notario o clúster de notarios que deberán llegar a consenso. Hasta que los nodos participantes en la transacción no han recibido la firma de los notarios no deberán dar la transacción por válida ni hacerla efectiva en su ledger.

Cuando la red ofrece un clúster de notarios estos deben acordar el algoritmo de consenso a usar: alta velocidad, baja velocidad, protocolo RAFT (https://raft.github.io/), … Corda permite que una misma red tenga varios clústeres, cada uno de ellos con su propio algoritmo de consenso, lo que posibilita:

- Elección del clúster más adecuado en función del propósito y las necesidades de la transacción.
- Balanceo de carga entre clústeres evitando una sobrecarga de trabajo y una ralentización de un clúster en concreto.
- Baja latencia al poder elegir el nodo que origina la transacción el clúster más cercano a él.

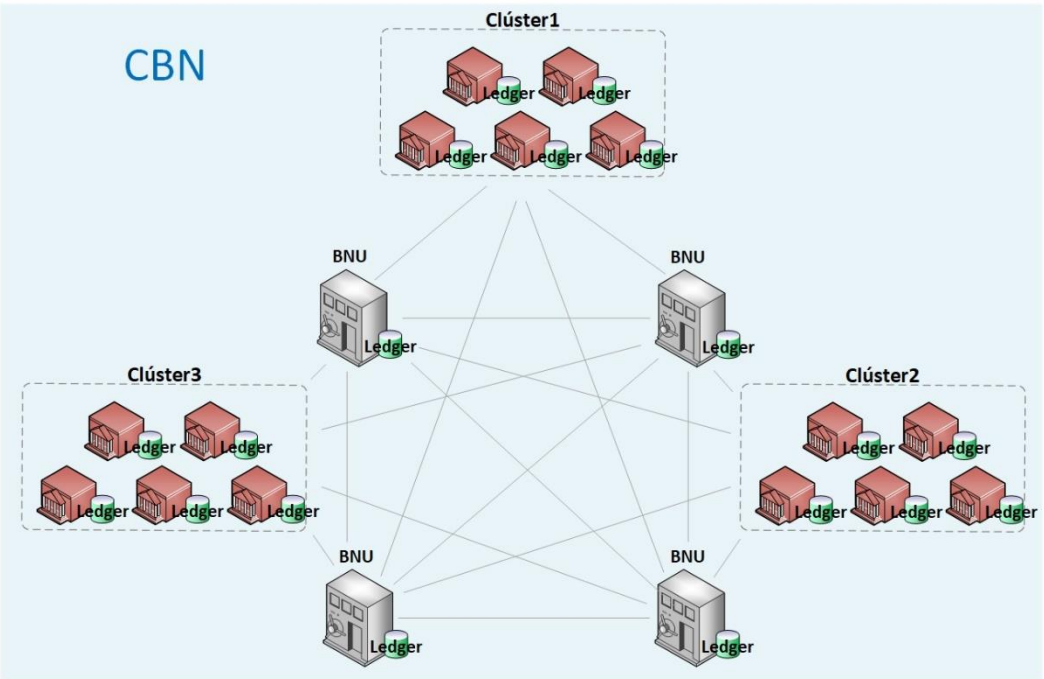


Figura 11. Clústeres de notarios

2.1.6 Aplicaciones Corda (CorDapp)

Una aplicación Corda (*CorDapp - Corda Distributed Application*) [13] es un fichero con extensión .jar (Java Archives) que contiene clases codificadas en lenguaje *Java* o *Kotlin* y que puede ser ejecutado en una máquina virtual de Java. Estas aplicaciones son distribuidas por los nodos de la de la plataforma Corda que harán uso de sus servicios.

Es una CorDapp quien permite a los nodos ejecutar transacciones y hacerlas efectivas sobre sus ledgers. Cada CorDapp define flujos que pueden ser inicializados por el propietario del nodo, o por cualquiera que tenga los permisos necesarios, a través de llamadas de procedimiento remoto (*RPC – Remote Procedure Call*).

Los tipos de clases que debe contener como mínimo una CorDapp son:

- **Flujos**: Definen los pasos que sigue la transacción durante su ejecución. Un flujo puede ser invocado remotamente mediante *RPC*.
- **Estados**: Definen los datos sobre los que un acuerdo es alcanzado. Se usan como entradas y salidas de las transacciones.
- **Contratos**: Verifican que la transacción cumple con las reglas acordadas y la validan para hacer efectivo una modificación de la ledger.

La figura 12 pretende mostrar la rutina que sigue un flujo de una CorDapp que es inicializado por una llamada externa RPC. En dicha figura se ven las interacciones del nodo con el resto de nodos que participan en la transacción, así como la interacción con el clúster de notarios.

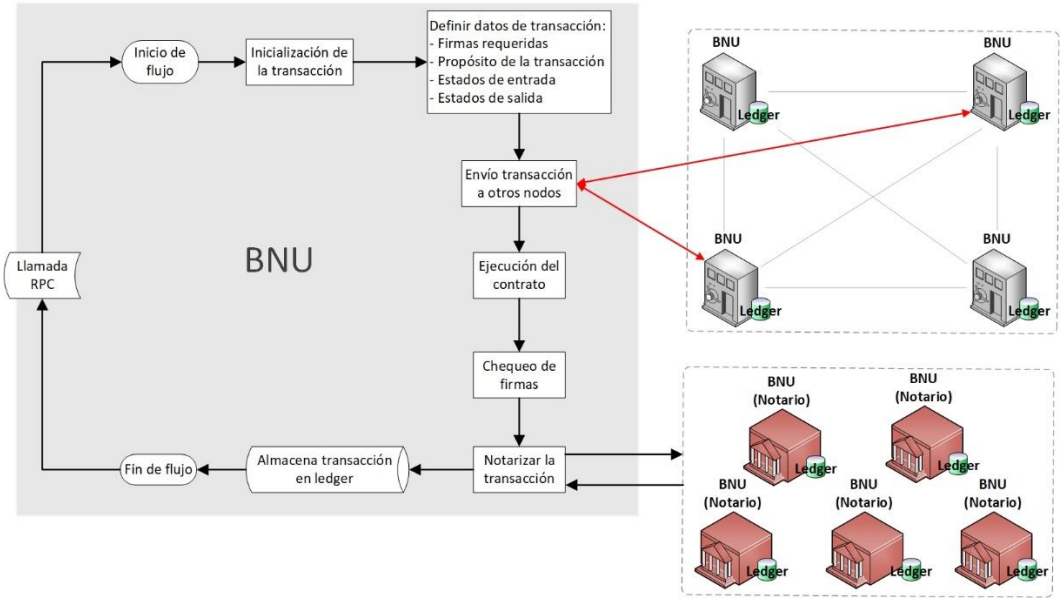


Figura 12. Flujo CorDapp

3. Análisis del sistema

Este apartado tiene como objetivo:

- Identificar todas las parte, metodologías y objetos, que pueden estar presentes durante la emisión, transferencia y depósito de cheques al portador, y la notación de cada uno de ellos dentro del esquema.
- Definir los requerimientos de seguridad a cumplir.
- Analizar la prueba de conocimiento nulo usada durante el proceso de transferencia.
- Detallar las consideraciones generales del esquema.
- Presentar la topología del sistema y la presencia de la red Corda en ella.
- Definir los protocolos de emisión, transferencia y depósito.

3.1 Intervinientes, metodologías y objetos

Para el correcto entendimiento de cada uno de los protocolos que se detallan en las próximas secciones se hace indispensable conocer los intervinientes, metodologías y objetos de los que se hace uso en cada uno de ellos, así como el acrónimo de cada uno de ellos. Por esto se hace necesario empezar el análisis del sistema con este apartado.

A continuación, se enumeran los diferentes términos y acrónimos a tener en cuenta:

- **Corda Business Network (*CBN*):** Red privada Corda que soporta el sistema de emisión, transferencia y depósito de cheques al portador. A ella estarán adheridos los bancos y notarios que participan en el esquema.
- **Business Network Governor (*BNG*):** Institución dentro de la red privada Corda que define el funcionamiento de la red y establece políticas y reglas que regirán el funcionamiento del sistema.
- **Business Network Operator (*BNO*):** Institución dentro de la red privada Corda que se encarga de la supervisión del sistema y de asegurar el cumplimiento de lo definido por el BNG.
- **Business Network User (*BNU*):** Los usuarios que operan dentro de la red privada Corda, también referidos como nodos. En el esquema serán los bancos y los notarios.
- **Cheque al portador (*Ch*):** Activo con valor nominal que es transferido entre usuarios, tantas veces como se desee, de forma anónima frente los bancos.
- **Id cheque (*ChId*):** Identificador único del cheque que es asignado en el momento de la emisión por el banco.
- **Clave secreta (*CS*):** Código alfanumérico definido por el usuario propietario de un cheque antes de la emisión o transferencia del mismo.

- **Teléfono (*T*):** Servirá como medio para identificar al propietario de un cheque y como medio para informar a los usuarios sobre el resultado de los procesos de emisión, transferencia o depósito de un cheque.
- **Referencia depósito (*R*):** Código alfanumérico con el que el banco depositario de un cheque identifica el depósito.
- **Poseedor del cheque (*P*):** Campo del cheque que almacena de manera encriptada el teléfono del propietario del cheque. P = SHA-256(T|CS).
- **Banco:** Nodo de la red Corda al que sus clientes pueden solicitar la emisión de un cheque, en este caso tendrá rol de banco emisor (*BE*), o el depósito, en este caso tendrá rol de banco depositario (*BD*).
- **Notario (*N*):** Nodo de confianza dentro de la red Corda que evita que una transacción no haga un doble uso de un estado. Permite alcanzar consenso de unicidad al esquema.
- **Tercera Parte Confiable (*TPC*):** Tercera parte de confianza para los usuarios que participan en la transferencia de un cheque. Será quien solicite la transferencia del cheque ante el banco emisor del mismo.
- **Cliente (*C*):** Cliente del banco emisor que solicita la emisión de un cheque al portador o cliente del banco depositario que solicita el depósito del cheque que posee. El esquema no ofrecerá anonimato a estos al ser su identidad conocida de antemano por su banco.
- **Usuario:** Sujeto que participa en la transferencia de un cheque al portador. Será usuario iniciador (*UI*) cuando sea el tenedor inicial del cheque y sea quien inicia la transferencia. Será usuario receptor (*UR*) cuando sea el sujeto al que se le entrega el cheque.
- **Transacción Corda (*Trx*):** Objeto de datos almacenado en la ledger del nodo correspondiente que fue originado en la red Corda como resultado de una emisión, transferencia o depósito.
- **Estado:** Objeto de información que puede ser usado por una transacción como datos de entrada (*I*) o como datos de salida generados durante la ejecución de la transacción (*O*).
- **Smart contract (SM):** Contrato inteligente que verifica que los estados usados en una transacción cumplen con las restricciones impuestas por el esquema en la emisión, transferencia y depósito.
- **Libro de contabilidad (Ledger):** Base de datos que almacena todas las transacciones y estados en las que participa el nodo.
- **Prueba de conocimiento nulo no-interactiva (*NIZKP – Non-Interactive Zero Knowledge Proof*):** Protocolo usado entre un usuario y la TPC que permite a este último comprobar que el usuario realmente posee lo que dice poseer.

- **Aplicación Corda (*CorDapp – Corda Distributed Application*):** Aplicación distribuida que deben ejecutar los nodos Corda adheridos al sistema.
- **AMQP (Advanced Message Queuing Protocol):** Protocolo de comunicaciones de la capa de aplicación usado por los nodos Corda durante la ejecución de la transacción.
- **Llamada a procedimiento remoto (*RPC - Remote Procedure Call*):** Protocolo que permite a un programa ejecutar subrutinas de otro programa de forma remota sin tener que conocer detalles de la red.
- **Seguridad de la capa de transporte (*TLS - Transport Layer Security*):** Protocolo criptográfico que permite asegurar las comunicaciones a través de red que puede ser pública**.**
- **Función hash (*H*):** Función unidireccional que haciendo uso de un algoritmo matemático transforma un bloque de datos de longitud variable en una cadena de caracteres de longitud finita. Se dice que es unidireccional porque a partir del resultado devuelto por la función no es posible obtener el bloque de datos sobre el que se aplicó la función.
- **SHA-256:** Función hash de la familia SHA-2 cuyo algoritmo es capaz de generar siempre una cadena de 64 caracteres hexadecimales.

- **Certificado X.509 [17]:** Certificado digital que usa el estándar de infraestructura de clave pública X.509 que garantiza que la clave pública pertenece a la entidad identificada y que la entidad posee la correspondiente clave privada. Permite establecer comunicaciones *TLS* e identificar al titular de la clave pública mediante su campo *Subject*. Este campo es expresado en notación *DN (Distinguished Name)* [18].

3.2 Requerimientos de seguridad

En este apartado se definen los requerimientos que debe cumplir el esquema para que pueda ser presentado como un sistema fiable y seguro de emisión, transferencia y depósito de cheques al portador.

Los requerimientos a cumplir son:

- **Infalsificable**

  El esquema debe garantizar que solo los bancos adheridos al sistema pueden emitir, transferir o depositar cheques. También, debe asegurar que ningún banco puede emitir, transferir o depositar cheques haciéndose pasar por otro. haciéndose pasar por otro. El cumplimiento de este requerimiento hará que la propiedad de no repudio sea satisfecha.

- **Anonimato**
  El esquema debe asegurar que la identidad de los usuarios que participan en una transferencia no sea revelada ni pueda ser conocida por ningún banco de la red. Este requerimiento no será satisfecho en los procesos de emisión ni de disposición al ser el usuario un cliente del banco en cuestión.

- **Transferibilidad**
  El esquema debe permitir que un cheque, durante su periodo de validez, sea transferido tantas veces como se desee sin necesidad de ser depositado en cada transferencia. En cada transferencia se debe garantizar que el usuario receptor del cheque recibe un cheque válido para volver a transferir o depositar en su banco. El anonimato de los usuarios que participan en cada transferencia debe asegurarse en todo momento.

- **Detección doble uso**
  El esquema será capaz de detectar aquellos intentos de un usuario de transferir o depositar un cheque que ya ha sido transferido o depositado previamente. Así mismo, se debe asegurar que un cheque no puede ser poseído por más de un usuario simultáneamente.

- **Equitatividad**
  El esquema debe garantizar que al finalizar cualquiera de los procesos ninguno de los sujetos que participan en él sale favorecido ni perjudicado, recibiendo todos ellos lo esperado. Lo esperado en cada proceso es:
  - Emisión: El banco emisor espera recibir en una cuenta interna suya el importe del cheque y el cliente que solicita el cheque espere recibir este.
  - Transferencia: El tenedor inicial del cheque espera saldar una deuda o pagar un servicio que el usuario al que se le entrega el cheque ha ofrecido u ofrecerá.
  - Depósito: El banco depositario espera recibir del banco emisor una transferencia de dinero que se corresponda con el importe del cheque, y el cliente del banco espera recibir en su cuenta un abono que se corresponda también con el importe del cheque.

  Tanto los bancos como la Tercera Parte Confiable deben asegurarse que siempre informan el resultado de un proceso y, además, deben definir mecanismos en los que poder recibir peticiones preguntado sobre el estado de un proceso.

- **Atomicidad**
  El esquema debe asegurar que las operaciones se finalizan satisfactoriamente o no se finalizan y que ante un fallo la operación no queda a medias ni origina una inconsistencia de datos.

3.3 ZKP (Zero Knowledge Proof)

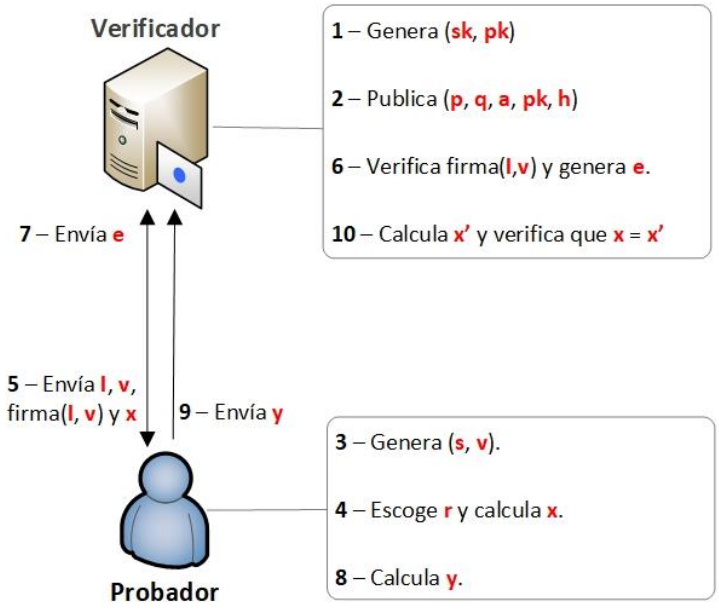
La prueba de conocimiento nula [14]-[16] permite al esquema asegurar el anonimato de los usuarios que participan en una transferencia, definiendo una rutina en la que esos usuarios puedan demostrar ante la Tercera Parte Confiable (*TPC*) su implicación en dicha transferencia sin revelar ningún secreto.

La prueba de conocimiento nulo se basa en el algoritmo de Schnorr [14]. Este algoritmo permite, a un usuario su autenticación ante un tercero sin tener que estar registrado ni tener que loguearse en él. También permite, a un usuario demostrar ante un tercero que realmente posee el documento o mensaje que dice poseer.

Según se quiera demostrar la autenticidad de un usuario o la posesión de un documento se llevará a cabo una prueba de conocimiento nulo interactiva o no interactiva respectivamente. En ambos casos el usuario tendrá el rol de probador y el tercero, sujeto ante el que el probador se autentica o al que quiere demostrar la posesión del documento, tendrá el rol de verificador. Este verificador tiene como obligación el facilitar o hacer llegar al probador una serie datos que este necesita para poder iniciar la prueba.

La autenticación del probador, prueba interactiva (*IZKP – Interactive Zero Knowledge Proof*), requiere que tanto probador como verificador estén disponibles durante la ejecución de la prueba ya que se produce un intercambio de datos durante la misma. En uno de estos intercambios, el verificador envía el probador un desafío o reto en base al cual este último tiene que dar una respuesta que será determinante para corroborar su autenticidad. La figura 13 pretende mostrar cómo sería el funcionamiento de esta prueba.

El esquema que presenta este documento no tiene como finalidad probar la autenticación de un usuario y sí probar que posee el documento, en este caso el cheque, que dice poseer. Esta demostración es posible llevarla a cabo con un modelo no interactivo de la prueba (*NIZKP – Non-Interactive Zero Knowledge Proof*), es decir, un modelo donde no se requiere que el verificador esté disponible durante la ejecución de la misma al necesitar el probador resolver desafío alguno. La forma que define la prueba para verificar un documento es en base a la firma del documento y al envío de esta, junto con el documento, al verificador. La figura 14 pretende mostrar cómo sería el funcionamiento de la prueba en este caso.



NOTACIÓN:

- **IZKP (Interactive Zero Knowledge Proof):** Prueba de conocimiento nulo o cero interactiva.
- **Probador:** Usuario que inicia la prueba y que tiene como objetivo demostrar su autenticidad.
- **Verificador:** Sujeto que publica los datos necesarios para que el probador inicie la prueba, que impone un desafio a este y que en función del resultado al desafío podrá verfiicar, o no, la autenticidad del probador.
- **sk:** Clave privada del verificador.
- **pk**: Clave pública del verificador.
- **p:** Número primo de orden de $\approx 2^{1024}$.
- **q:** Número primo divisor de p-1 ($\approx 2^{160}$).
- **a:** $1 < a < p$ y $a^q \equiv 1 \pmod p$.
- **h:** Función hash a usar.
- **s:** Clave privada del probador $0 < s <= q$
- **v:** Clave pública del probador. Cálculo: $v = a^{-s} \pmod p = a^{q-s} \pmod p$
- **r:** Número aleatorio tal que: $0 < r < q$.
- **x** $= a^r \pmod p$
- **I:** Identificación del usuario.
- **e (Desafio):** Número aleatorio $0 < e < 2^t$
- **t (Número de seguridad)**: Varia en función de la aplicación del algoritmo.
- **y** $= r + se \pmod q$
- **x'** $= a^y v^e \pmod p$

Figura 13. IZKP (Prueba de conocimiento nula interactiva)

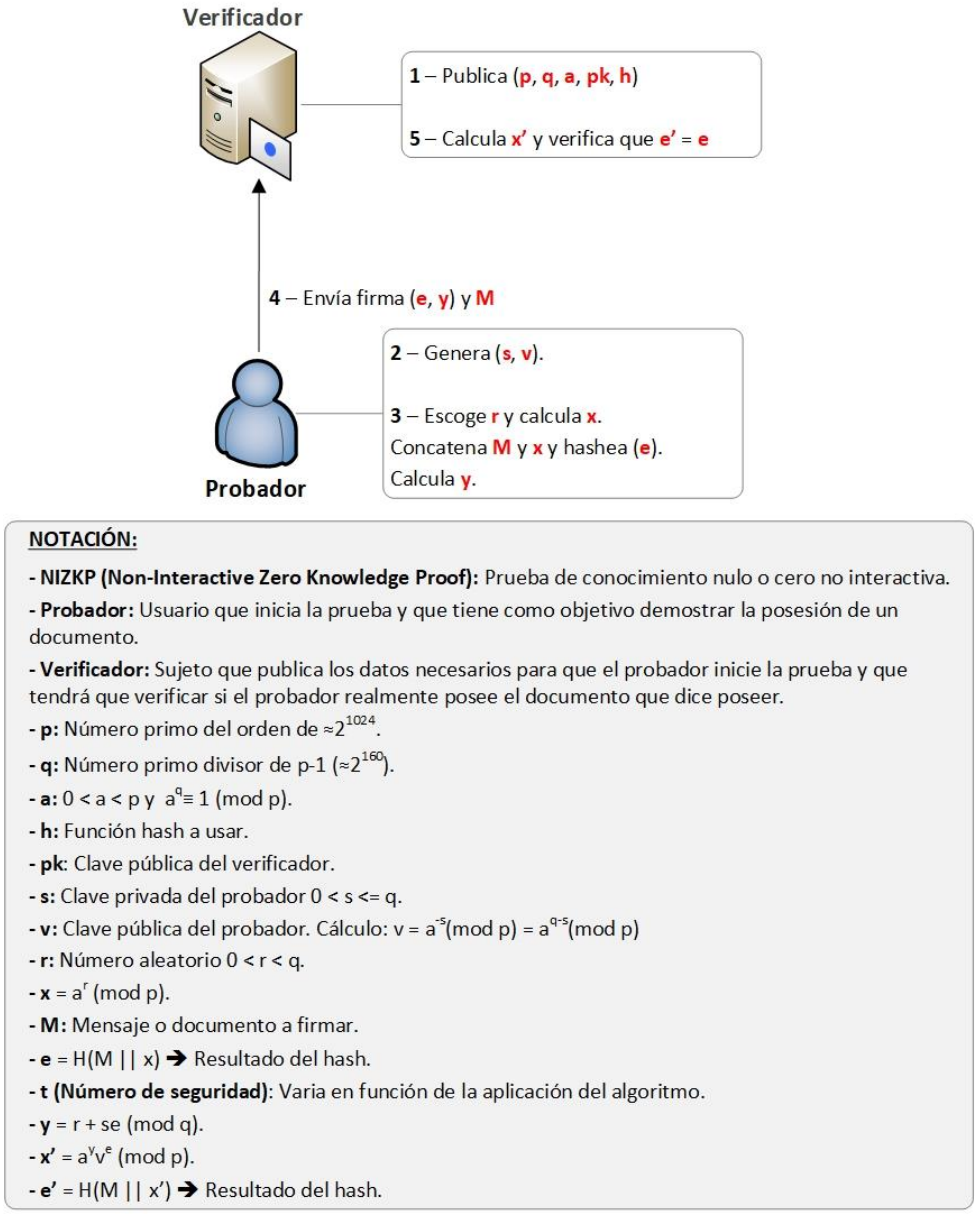


Figura 14. NIZKP (Prueba de conocimiento nula no interactiva)

3.4 Consideraciones generales

Dentro del análisis de todo sistema es necesario establecer al inicio unos requisitos que sirvan de guía durante la fase de desarrollo. En este caso se han definido una serie de premisas para cada una de las partes y objetos que pueden aparecer en los diferentes protocolos.

La tarea de definición de requisitos no acaba en la fase de análisis. Durante la fase de desarrollo pueden presentarse circunstancias, relacionadas por ejemplo con la tecnología empleada, que requieran añadir, eliminar o modificar algún requisito de los definidos a continuación. También, pueden presentarse nuevas consideraciones que no se han tenido en cuenta en este punto del trabajo y que obligarán a ser añadidas más

tarde siempre y cuando se compruebe que afectan o mejoran el funcionamiento del sistema.

Las consideraciones iniciales han sido hechas sobre:

- Cheque al portador.
- Usuario.
- Tercera Parte Confiable.
- Banco.
- Notario.
- Red Corda.
- Transacción Corda.
- Smart contract o contrato.

Aunque no es objeto de este proyecto llevar a cabo una implementación completa del esquema, sí que las consideraciones indicadas a continuación sería necesario tenerlas en cuenta para una futura implementación.

Cheque al portador:

a. Son emitidos por un banco a petición de un cliente suyo.
b. La emisión de un cheque conlleva un adeudo en la cuenta del cliente.
c. Contiene:
    - Banco emisor del cheque.
    - Importe y divisa del cheque.
    - Periodo de validez.
    - P (Poseedor).
    - Id único.
    - Tipo de cheque.
d. Un identificador único es asignado en la emisión del cheque. No varía durante la vida útil del cheque.
e. Cada cheque contiene un campo Poseedor que varía con cada transferencia y depósito.
f. El campo Poseedor almacena el resultado devuelto por una función SHA-256 sobre el teléfono y la clave secreta del tenedor del cheque.
g. El objetivo de *P* es ofrecer anonimato a los usuarios que participan en la transferencia al no ser posible obtener el teléfono del tenedor del cheque a partir de este campo.

h. La clave secreta es definida en la emisión por el cliente del banco emisor y en la transferencia por el usuario que recibe el cheque.

i. La clave secreta aporta seguridad a la transferencia y depósito de un cheque evitando que un tercero, que sin ser tenedor del cheque conoce sus datos, pueda transferirlo o depositarlo sin el conocimiento de esta clave.

j. Se usa el teléfono como medio para la identificación del usuario y con el objetivo de poderle informar sobre el resultado de una emisión, transferencia o depósito.

k. En la transferencia de un cheque tanto la clave secreta como el teléfono forman parte del mensaje a probar en la prueba de conocimiento nulo.

l. Dentro del periodo de validez puede ser transferido tantas veces como se desee antes de ser depositado.

m. El periodo de validez no puede ser modificado.

n. En caso de caducidad el importe del cheque será abonado en la cuenta del cliente que solicitó su emisión. El tenedor del cheque en ese momento pierde la posibilidad de depositarlo.

o. El depósito de un cheque debe de hacerse dentro de su periodo de validez.

p. El depósito de un cheque conlleva un abono en la cuenta del cliente.

Usuario:

a. Como cliente de un banco puede solicitar la emisión o el depósito de un cheque al portador.

b. Como cliente del banco emisor:

   - Al solicitar la emisión de un cheque debe generar una clave secreta e informarla a su banco para que genere *P*.
   - Al solicitar la emisión de un cheque se le adeudará el importe del mismo de su cuenta.
   - Debe almacenar en un sitio seguro la clave secreta que definió y el cheque que le emitió su banco.
   - Una vez se haya completado la emisión, recibirá notificación en su teléfono informándole de un adeudo en su cuenta como consecuencia de la emisión.

c. Como cliente del banco depositario:

   - Solo podrá solicitar el depósito de un cheque que le fue transferido a su número de teléfono, siendo este el que el banco tiene asociado internamente a su cuenta.

- Al solicitar el depósito de un cheque debe proporcionar a su banco la clave secreta del cheque.
- Si la operación de depósito se terminó de ejecutar correctamente el importe del cheque es abonado en su cuenta
- Una vez se haya completado el depósito, recibirá una notificación en su teléfono informándole de un abono en su cuenta como consecuencia del depósito.

d. Todo usuario que posea un cheque debe almacenarlo en un sitio seguro junto con la clave secreta que definió para él.

e. En la transferencia de un cheque puede ser el iniciador de la transferencia (en ese momento tenedor del cheque) o el receptor de la transferencia (tenedor del cheque una vez haya finalizado la transferencia).

f. Como iniciador de una transferencia:
    - Su objetivo es entregar el cheque que posee a otro usuario.
    - Tiene rol de probador en la prueba de conocimiento nulo (*NIZKP*) y como tal la inicia.
    - El objetivo de iniciar la prueba de conocimiento nulo es demostrar que posee un cheque del que conoce la clave secreta y el teléfono que dieron origen a *P*.
    - En la prueba de conocimiento nulo debe generar el mensaje a verificar con el siguiente contenido:
        - El cheque a transferir.
        - Su teléfono.
        - La clave secreta que definió en la emisión del cheque o cuando lo recibió en una transferencia.
        - El teléfono del usuario al que pretende entregar el cheque.
    - Debe entregar el cheque al destinatario de la transferencia.
    - Una vez se haya completado la transferencia, recibirá una notificación en su teléfono informándole de que el cheque ha sido transferido y que ya no es el tenedor del mismo.

g. Como receptor de una transferencia:
    - Su intención es recibir el cheque que posee otro usuario al final de la transferencia
    - Tiene rol de probador en la prueba de conocimiento nulo (*NIZKP*) y como tal la inicia.

- El objetivo de iniciar la prueba de conocimiento nulo es demostrar que ha recibido un cheque y que al final de la transferencia será el tenedor del mismo.
- En la prueba de conocimiento nulo debe generar el mensaje a verificar con el siguiente contenido:
  - El cheque a transferir.
  - La clave secreta sobre la que se calculará el *P* del cheque transferido.
  - Su teléfono.
- Debe almacenar en un sitio seguro la clave secreta que definió y el cheque que le emitió su banco.
- Una vez se haya completado la transferencia, recibirá una notificación en su teléfono informándole que es el nuevo tenedor.

h. Como tenedor de un cheque, que todavía no ha transferido, puede solicitar el depósito en su banco.

Tercera Parte Confiable:

a. Tercera parte de confianza para todos los usuarios que decidan participar en la transferencia de cheques.

b. Ejerce de frontend entre los usuarios que acuerdan la transferencia de un cheque y el banco emisor de este.

c. No lleva a cabo ningún registro de usuarios.

d. Tiene rol de verificador en la prueba de conocimiento nulo (*NIZKP*) y como tal espera a que un probador le envíe el contenido a verificar.

e. Sabrá que una prueba de conocimiento nulo la inició el usuario iniciador de la transferencia cuando reciba la firma de un mensaje y un mensaje formado por:
   - El cheque a transferir.
   - Un primer teléfono que se correspondería con el del usuario iniciador.
   - Una clave secreta que sería la que definió en su momento el usuario iniciador.
   - Un segundo teléfono que se correspondería con el del usuario al que se le quiere entregar el cheque (usuario receptor).

f. Sabrá que una prueba de conocimiento nulo la inició el usuario receptor de la transferencia cuando reciba la firma de un mensaje y un mensaje formado por:
   - El cheque a transferir.

- Un teléfono que se correspondería con el del usuario receptor.
- Una clave secreta que sería la que el usuario receptor define para el cálculo del nuevo *P* del cheque.

g. Ante una prueba de conocimiento nulo iniciada por el usuario iniciador debe:
   - Verificar la firma del mensaje con el mensaje recibido.
   - Comprobar que el *P* del cheque es igual al resultado obtenido de una función SHA-256 sobre el primer teléfono y la clave secreta recibidos en el mensaje.
   - Esperar a que el usuario receptor de la transferencia le envíe el mensaje a verificar.

h. Ante una prueba de conocimiento nulo iniciada por el usuario receptor debe:
   - Verificar la firma del mensaje con el mensaje recibido.
   - Verificar que ya recibió de un usuario iniciador un cheque a transferir que se corresponde con el que informó el usuario receptor en su mensaje.
   - Verificar que el segundo teléfono que informó el usuario iniciador en su mensaje se corresponde con el teléfono que informo el usuario receptor en su mensaje.

i. Calcula *P* para el cheque a partir del teléfono y la clave secreta que informó el usuario receptor en su mensaje.

j. Solicitar al banco emisor del cheque su transferencia, informándole el *P* calculado en el punto anterior.

k. No debe almacenar internamente las claves secretas que los usuarios definen durante una transferencia.

l. Una vez se haya completado la transferencia, debe informar al usuario iniciador que ha dejado de ser el propietario del cheque y al usuario receptor informarle que es el nuevo propietario y hacerle llegar el cheque cuyo *P* fue modificado con el calculado en el punto i.

m. En caso de no completarse la transferencia, también debe informar a ambos usuarios sobre el suceso.

Banco:

a. Para su adhesión a la red Corda debe preguntar al *BNG* de la misma sobre las reglas que rigen el funcionamiento de la misma.

b. Una vez aceptadas las reglas de la red debe solicitar al *BNO* de la red el acceso a la misma.

c. Será un nodo de la red Corda que tendrá su propia ledger en la que almacenará las transacciones Corda.

d. Validará todas las transacciones Corda en las que participa. Participa, por tanto, en alcanzar un consenso de validez de cada transacción.

e. Será identificado de manera única dentro de la red gracias al campo *Subject* del certificado que le emitió el *BNO*.

f. El campo *Subject* informará:

   - Organization (*O*).
   - Locality (*L*).
   - Country (*C*).

g. Las obligaciones que adquiere un banco durante el proceso de emisión del cheque son:

   - Calcula el *P* a partir del teléfono que el banco tiene asociado al cliente y de la clave secreta que el cliente define.
   - Generar cheque con un identificador único y con el *P* previamente calculado.
   - Adeudar de la cuenta del cliente el importe del cheque y depositarlo en una cuenta técnica interna.
   - Iniciar transacción Corda cuyo objetivo es generar un estado que contenga el cheque.
   - Entregar el cheque al cliente una vez ha finalizado la emisión del cheque o informarle un error durante el proceso.

h. Las obligaciones que adquiere un banco durante el proceso de transferencia del cheque son:

   - Iniciar transacción Corda cuyo objetivo es consumir el estado que se corresponde con el cheque a transferir y, generar un nuevo estado que contenga el cheque con el nuevo *P* que le informó la Tercera Parte Confiable.
   - Entregar el cheque a la Tercera Parte Confiable una vez ha finalizado la transferencia del cheque o informarle un error durante el proceso.

i. Las obligaciones que adquiere un banco durante el proceso de depósito de un cheque varían en función de su papel como emisor o como depositario del cheque.

   - Cuando actúa como emisor del cheque:
     - Iniciar transacción Corda cuyo objetivo es consumir el estado que se corresponde con el cheque a depositar y, generar un nuevo estado que informe el banco depositario y el cheque con el nuevo con el nuevo valor *P* que le informó el banco depositario.

        - Entregar el cheque al banco depositario como confirmación de que el depósito es posible o informarle un error durante el proceso.

    - Cuando actúa como depositario del cheque:
        - Recuperar el teléfono que tiene asociado al cliente que solicita el depósito.
        - Calcular valor del campo depositario a partir del teléfono recuperado en el punto anterior y de la clave secreta recibida de su cliente.
        - Verificar que el *P* del cheque que solicita depositar su cliente se corresponde con el *P* anteriormente calculado.
        - Generar nuevo *P* a partir de una referencia que asigna a la operación de depósito.
        - Solicitar depósito del cheque al banco emisor, informándole también el *P* calculado en el punto anterior.
        - Iniciar transacción Corda, que haga efectivo el depósito del cheque en la cuenta del cliente, una vez ha recibido la respuesta del banco emisor a su solicitud de depósito. Esta transacción consume el estado generado por la transacción anterior y no genera ninguno nuevo

j. Una vez efectuado el depósito de un cheque la liquidación del importe del cheque entre los bancos involucrados debe de hacerse a través de alguno de los sistemas de compensación existentes: TARGET 2, SEPA, …

Notario:

a. Conoce de antemano el funcionamiento del sistema con lo que no necesita contactar con el *BNG*.

b. Debe solicitar al *BNO* de la red el acceso a la misma.

c. Podrá ser un clúster de notarios para ofrecer una mayor seguridad a la red y evitar que se comprometa el sistema en caso de ser atacado.

d. Será un nodo de la red que no establece comunicaciones con ningún elemento externo a la misma.

e. Será un nodo de la red Corda que tendrá su propia ledger en la que almacenará las transacciones Corda que valida.

f. Es un nodo de confianza para los bancos de la red.

g. Se encarga de ofrecer consenso de unicidad al esquema, evitando que las transacciones hagan usos de estados previamente usados por otras.

h. Validará todas las transacciones Corda en las que participa. Participa, por tanto, en alcanzar un consenso de validez de cada transacción.

Red Corda:

a. Red privada formada por:
   - Un *BNG*.
   - Un *BNO*.
   - Nodos interconectados entre sí que pueden ser bancos o notarios.

b. Los principales requerimientos técnicos de cualquier nodo de la red (banco o notario) serán:
   - Desplegar el software de nodo Corda en un servidor dedicado.
   - Instalar la *CorDapp* en el nodo.

c. A efectos prácticos el *BNG* no requiere el despliegue de un nodo.

d. El *BNG* y el *BNO* pueden recaer sobre el mismo organismo.

e. La caída de un nodo no impide el funcionamiento de la red salvo en aquellos casos en los que una transacción requiere de dicho nodo.

Transacción Corda:

a. Es iniciada por cualquier banco de la red

b. La transacción ejecutada en los procesos de emisión y transferencia solo involucran al banco emisor del cheque.

c. La transacción ejecutada en el proceso de depósito y la que hace efectiva dicho depósito, es decir el ingreso en la cuenta del cliente, involucran tanto al banco emisor del cheque como al banco depositario.

d. Las transacciones ejecutadas en los procesos de emisión, transferencia y depósito son iniciadas por el banco emisor del cheque.

e. La transacción efectuada para hacer efectivo el depósito del cheque en la cuenta del cliente es iniciada por el banco depositario del cheque.

f. En las transacciones participan estados que contienen los datos del cheque.

g. En función del proceso la transacción hará uso, o no, de estados previos y generará, o no, nuevos estados.
   - Proceso de emisión: No usa (consume) ningún estado previo y genera uno con los datos del cheque a emitir.
   - Proceso de transferencia: Consume un estado previo que se corresponde con el cheque a transferir y genera uno nuevo que se corresponde con el cheque a entregar. Este último solo varía el *P* con respecto al consumido.
   - Proceso de depósito: Consume un estado previo que se corresponde con el cheque a depositar y genera uno nuevo que se corresponde con el cheque a hacer efectivo en la cuenta del cliente. Este último solo varía el *P* con respecto al consumido. La transacción que hace efectivo el ingreso

del cheque en la cuenta del cliente consume el estado generado por la transacción anterior y no genera uno nuevo.

h. El notario garantiza que ninguna transacción hace uso de un estado previamente consumido. El intento de consumir un estado ya consumido provoca el rechazo de la transacción y la notificación correspondiente.

i. Se considera que la emisión, la transferencia y el depósito han terminado cuando la ejecución de la transacción ha terminado.

j. Las transacciones requieren la firma de los bancos que participan en ella y del notario para alcanzar un consenso de validez.
   - Proceso de emisión: Requiere firma del banco emisor y del notario.
   - Proceso de transferencia: Requiere firma del banco emisor y del notario.
   - Proceso de depósito: Requiere firma del banco emisor, del banco depositario y del notario

k. Con la firma del notario se garantiza que no se ha producido un doble gasto del estado consumido. Se proporciona, por tanto, consenso de unicidad.

l. La ejecución de las transacciones de los procesos de emisión y transferencia producen cambios sobre las ledgers del banco emisor y del notario.

m. La ejecución de la transacción del proceso de depósito produce cambios sobre las ledgers del banco emisor, del banco depositario y del notario.

Smart contracts:

a. Corda se refiere a ellos simplemente como contratos.

b. Durante el proceso de emisión verifica que el cheque cumple con lo requerido, abortando su emisión en caso de ser necesario.

c. Durante los procesos de emisión y transferencia verifica que ningún banco que no haya sido el emisor del cheque autoriza su transferencia o depósito.

d. Durante la ejecución de la transacción que hace efectivo el ingreso del cheque verifica que ningún banco que no haya sido el depositario del cheque autoriza el ingreso.

3.5 Topología del sistema

Este apartado ofrece una presentación en conjunto de todos los intervinientes que participarían en el esquema, aunque no todos ellos estén presentes en cada uno de los procesos. La figura 15 pretende mostrar las comunicaciones entre las distintas partes, la tecnología usada en aquellos casos en los que algún interviniente de la comunicación es un elemento de la red Corda o la Tercera Parte Confiable e, identificar ciertos servicios que deben ofrecer tanto el banco emisor como el depositario para que se puedan llevar a cabo las emisión, transferencia y depósito de un cheque.

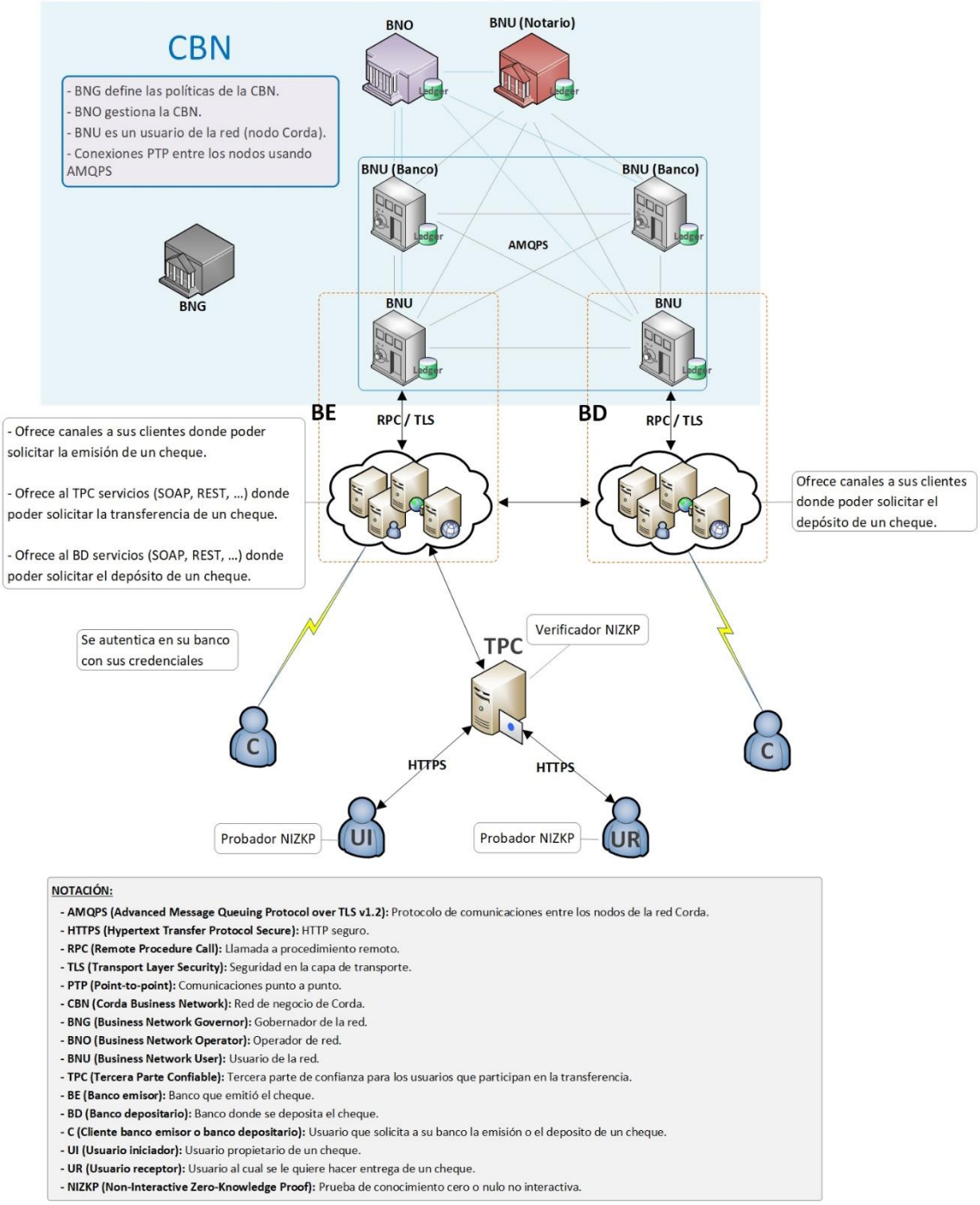


Figura 15. Topología del sistema

Analizando la figura 15 se observa que no es objetivo de este esquema el definir:

- Los mecanismos de autenticación de los clientes antes sus bancos.
- Los canales que los bancos ofrecen a sus clientes para la emisión y depósito de un cheque.
- Los servicios que un banco emisor debe ofrecer a la Tercera Parte Confiable para que esta solicite la transferencia de un cheque.

- Los servicios que un banco emisor debe ofrecer al resto de banco para que estos le soliciten iniciar el depósito de un cheque.

Tampoco es objetivo del esquema a estudio el definir las tecnologías y protocolos usados en las comunicaciones entre bancos y cliente o entre banco y la Tercera Parte Confiable sí que se da por hecho que estos canales deberán ofrecer las garantías mínimas de seguridad para que ningún ataque pueda comprometer el sistema en general.

Aunque hasta ahora se había hablado siempre del banco emisor y del banco depositario como un nodo Corda, en la figura 15 se hace una aproximación más real a cuáles serían las partes de las que constan. Así, el banco tendrá un conjunto de servicios y sistemas corporativos, quedando estos fuera del análisis por este proyecto, y un nodo Corda que permitirá a esos servicios y sistemas solicitar la emisión y depósito de un cheque a través de llamadas de procedimiento remoto (RPC – Remote Procedure Call) sobre canales seguros (TLS – Transport Layer Security).

A partir de la topología vista conviene destacar o hacer ciertas recomendaciones sobre sobre la instalación y el despliegue de los nodos Corda:

- Se recomienda una correcta implantación de mecanismos de redundancia y replicación de todos los intervinientes en el sistema que por su rol puedan recibir una alta demanda de trabajo (*BNG* y *BNO*).
- El *BNG* tendrá el derecho y el deber de modificar las bases legales y regulatorias que rigen la *CBN* para contemplar el crecimiento de esta sin que ello suponga una pérdida de calidad del sistema.
- La *CBN* tendrá que ser capaz de soportar tantos nodos como bancos deseen adherirse al sistema.
- La *CBN* tendrá que ser capaz de soportar un aumento del número de notarios presentes en la red según vaya creciendo el número de bancos adheridos.
- El *BNO* tendrá que poder aumentar su capacidad de gestión y control sobre la *CBN* a medida que el número de bancos y notarios aumenta.

También conviene destacar que un aumento de las transferencias de cheques entre usuarios implica una mayor demanda de trabajo a la Tercera Parte Confiable. Esto implica que llegado cierto punto se haga necesario implantar sistemas de redundancia y de balanceo de carga para asegurar la disponibilidad del sistema en todo momento.

Así mismo, conviene detectar e informar de que el notario de la red supone el punto más débil del sistema. El hecho de verse comprometido afectará no solo a la disponibilidad del sistema sino también, a las tareas en las que participa para alcanzar consenso de validez y unicidad de las transacciones.

Por último, conviene destacar que la prueba de conocimiento nulo entre usuarios y la Tercera Parte Confiable debe ser llevada a cabo sobre un canal de comunicaciones seguro que no permita a un tercero acceder a los mensajes enviados.

3.6 Protocolo de emisión de un cheque

Este apartado tiene como objeto definir el protocolo de emisión de un cheque que es solicitado por un cliente a su banco por un importe y validez determinado. La ejecución del protocolo conlleva que el cliente reciba el cheque solicitado y que el banco haga el adeudo correspondiente sobre la cuenta del cliente.

Las partes que intervienen en este protocolo son son:

- Banco emisor (*BE*).
- Cliente del banco emisor (*C*).
- Notario (*N*).

3.6.1 Pasos del protocolo

Al no definir el esquema el canal a través del cual *C* puede solicitar la emisión de un *Ch* a su banco, no se considera como paso del protocolo la autenticación de *C* ante su banco. Se parte de la certeza que *BE* ya ha identificado a su *C* y que este ya está en disposición de solicitar la emisión de un cheque.

Paso 1: *C* genera una *CS* que le permitirá una vez ha obtenido *Ch* poder transferirlo.

Paso 2: *C* solicita a *BE* la emisión de un *Ch* y le proporciona la *CS* que generó en el paso 1.

Paso 3: *BE* hace las comprobaciones y retenciones de dinero necesarias sobre la cuenta de *C* y recupera el *T* que tiene asociado a él. Calcula *P* (función SHA-256 sobre *T* y *CS*) y genera *Ch* que contiene: identificador único (*ChId*), *P* previamente calculado, tipo del cheque e importe, divisa y periodo de validez que *C* definió al hacer la solicitud.

Paso 4: *BE* invoca a su nodo Corda para iniciar una *Trx* que tenga como finalidad generar un estado (*O*) con los datos de *Ch*. La ejecución de *Trx* supone que la CorDapp tiene que:

- Ejecutar *SC* que valide O.
- Almacenar en su *ledger O* y *Trx*.

La CorDapp de *N* tiene que validar la *Trx* y devolver su firma como comprobante de que *Trx* es válida para él.

Paso 5: *BE* transfiere el importe del *Ch* que retuvo a *C* en el paso 3 a una cuenta técnica interna donde permanecerá hasta que *Ch* sea depositado o cumpla su periodo de validez.

Paso 6: *BE* hace llegar a su *C* el *Ch*.

La figura 16 muestra de forma gráfica los distintos pasos de los que consta el protocolo y que han sido detallados en esta sección.

Conviene tener claro que el campo *P* del *Ch* será quien permita concluir quién es el propietario del *Ch* al estar generado este campo en base a una *CS* que solo *C* debe conocer.

Para el esquema el fin de la emisión de un *Ch* acaba con el paso 4 ya que se pueden implementar mecanismos que hagan posible volver a ejecutar los pasos 5 y 6 sin solicitar la intervención de *C*.

3.6.2 Incidencias o motivos de rechazo en la emisión

Este apartado hace constar los posibles motivos que durante la emisión de un *Ch* pudiesen surgir y provocasen que el proceso de emisión no se completase.

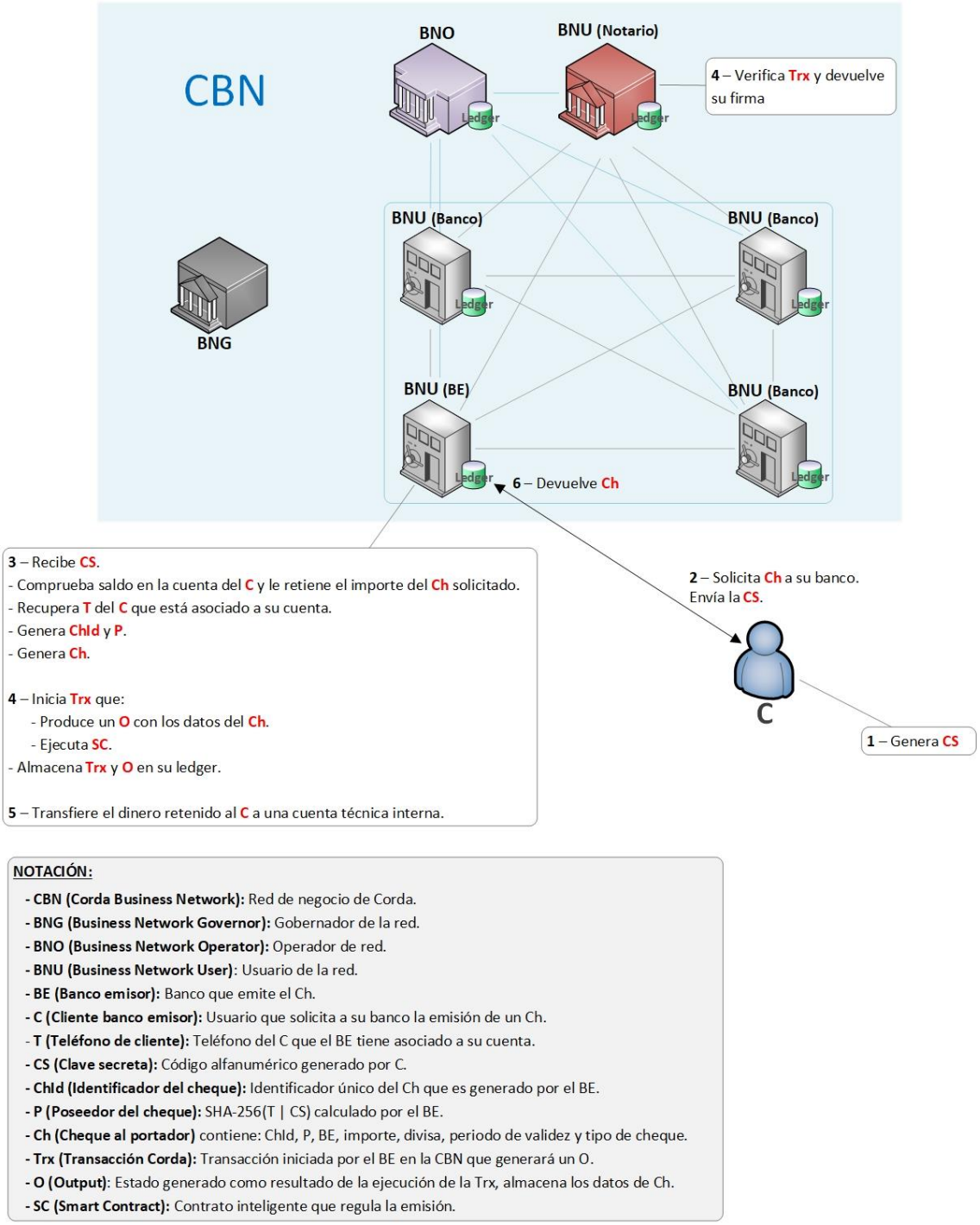
El *C* no tiene saldo suficiente en cuenta.
No se genera *Ch*.

El *BE* no tiene asociado ningún *T* a la cuenta de *C.*
No se genera *Ch*. *BE* solicita a *C* que informe un *T*.

El *BE* no es capaz de ejecutar función SHA-256 sobre *T* y *CS.*
No se genera *Ch*. *BE* informa del error a *C*.

El *BE* recibe error durante la ejecución de *Trx.*
No se genera *Ch*. *BE* informa del error a *C* y el importe que le fue retenido es liberado.

El *C* no recibe *Ch* de *BE* ni respuesta sobre el resultado de la emisión.
C está en disposición de preguntar a *BE* sobre el proceso de emisión tantas veces como desee y de recibir el *Ch* en caso de haberse emitido.



NOTACIÓN:

- **CBN (Corda Business Network):** Red de negocio de Corda.
- **BNG (Business Network Governor):** Gobernador de la red.
- **BNO (Business Network Operator):** Operador de red.
- **BNU (Business Network User)**: Usuario de la red.
- **BE (Banco emisor):** Banco que emite el Ch.
- **C (Cliente banco emisor):** Usuario que solicita a su banco la emisión de un Ch.
- **T (Teléfono de cliente):** Teléfono del C que el BE tiene asociado a su cuenta.
- **CS (Clave secreta):** Código alfanumérico generado por C.
- **ChId (Identificador del cheque):** Identificador único del Ch que es generado por el BE.
- **P (Poseedor del cheque):** SHA-256(T | CS) calculado por el BE.
- **Ch (Cheque al portador)** contiene: ChId, P, BE, importe, divisa, periodo de validez y tipo de cheque.
- **Trx (Transacción Corda):** Transacción iniciada por el BE en la CBN que generará un O.
- **O (Output)**: Estado generado como resultado de la ejecución de la Trx, almacena los datos de Ch.
- **SC (Smart Contract):** Contrato inteligente que regula la emisión.

Figura 16. Protocolo de emisión

3.7 Protocolo de transferencia de un cheque

Este apartado tiene como objeto definir el protocolo de transferencia de un cheque cuando dos usuarios, uno de los cuales posee un cheque, acuerdan su transferencia. La ejecución del protocolo conlleva que la propiedad del cheque pase de un usuario a otro y que ambos queden conformes con la transferencia al terminar el proceso.

Las partes que intervienen en este protocolo son:

- Usuario propietario de un cheque, será el iniciador de la transferencia (*Ui*).
- Usuario al que se quiere entregar el cheque, será el receptor de la transferencia (*Ur*).
- Agente Intermediario (*AI*).
- Banco emisor (*Bem*).
- Notario (*N*).

3.7.1 Pasos del protocolo

No queda recogido dentro de ninguno de los pasos del protocolo el motivo por el que *UI* y *UR* acuerdan la transferencia de un *Ch* ni tampoco el servicio que se pretende satisfacer con esta transferencia.

Paso 1: *UI*, como propietario del *Ch* a transferir, inicia la *NIZKP* con la que pretende probar que posee un *Ch* y que conoce el *T* y la *CS* que dan origen a *P*. *UI* también pretende probar que conoce el *T* del *UR* (se anota el *T* del *UR* como *T'* para evitar confusiones con el *T* de *UI*) al que quiere su *Ch*.

Paso 2: El *TPC* verifica la *NIZKP* iniciada por *UI* y comprueba que el *P* del *Ch* es igual al resultado obtenido del cálculo de la función SHA-256 sobre *T* y *CS.*

Paso 3: *UI* entrega el *Ch* a *UR*.

Paso 4: *UR* genera una *CS* (se anota la *CS* de *UR* como *CS'* para evitar confusiones con la *CS* del *UI*) con la que pretende asegurar que en caso de finalizar la transferencia él será el nuevo propietario del *Ch*.

Paso 5: *UR* inicia la *NIZKP* con la que pretende probar que un usuario ha solicitado transferirle un *Ch*. Adjunta *T'* y *CS'* como parte del mensaje a probar.

Paso 6: El *TPC* verifica la *NIZKP* iniciada por *UR.* Comprueba que el *Ch* que recibió de *UI* y el que recibió de *UR* son iguales y, también comprueba que el *T'* que recibió de *UI* y el que recibió de *UR* son iguales. Genera *P* en base a *T'* y a *CS'* (se anota el

P aquí calculado como *P'* para evitar confusiones con el *P* del *Ch* que se solicita transferir).

Paso 7: El *TPC* solicita al *BE* la transferencia de *Ch* y le envía *P'*.

Paso 8: *BE* recibe *Ch* y *P'*.

Paso 9: *BE* invoca a su nodo Corda para iniciar una *Trx* que tenga como finalidad consumir un *I* y generar un *O* con los datos de *Ch.* La ejecución de *Trx* supone que la CorDapp tiene que:

- Recuperar de su *ledger* el *I* que se corresponda con el *Ch* a transferir.
- Actualizar el *P* del *Ch* con el *P'* recibido.
- Generar un *O* a partir del *Ch*.
- Hacer que el *SC* verifique tanto *I* como *O*.
- Marcar el *I* como consumido.
- Almacenar en su *ledger O* y *Trx*.

La CorDapp de *N* tiene que validar la *Trx* y devolver su firma como comprobante de que *Trx* es válida para él y que *I* no había sido consumido previamente.

Paso 10: *BE* devuelve el *Ch* al *TPC*.

Paso 11: *TPC* informa al *UI* la transferencia del *Ch*.

Paso 12: *TPC* informa a *UR* que es el nuevo tenedor del *Ch* y como tal se lo entrega.

La figura 16 muestra de forma gráfica los distintos pasos de los que consta el protocolo y que han sido detallados en esta sección.

La transferencia de un cheque no supone un movimiento de fondos entre el *UI* y el *UR*. El *TPC* no almacena ninguna de las *CS* que *UI* y *UR* le informan durante la *NIZKP*.

Este protocolo asegura el anonimato de los usuarios que participan en la transacción (*UI* y *UR*) ante el *BE* no almacenando en el *Ch* ningún campo que permita identificarles. Así mismo, gracias a la *CS* que el *UI* definió antes de ser el propietario del *Ch*, se asegura que ningún otro usuario está en disposición de transferir el *Ch*.

Para el esquema el fin de la transferencia de *Ch* acaba con el paso 9 ya que se pueden implementar mecanismos que hagan posible volver a ejecutar los pasos siguientes sin necesidad de que *UI* y *UR* intervengan.

3.7.2 Incidencias o motivos de rechazo en la transferencia

Este apartado hace constar los posibles motivos que durante la transferencia de un *Ch* pudiesen surgir y provocasen que el proceso de transferencia no se completase.

La verificación que hace el *TPC* del *NIZKP* iniciado por *UI* no es satisfactoria.
El *TPC* informa el suceso al *UI* y no solicita al *BE* la transferencia de *Ch*.

El *TPC* no es capaz de generar un *P* en base al *T* y al *CS* que el *UI* le proporcionó.
El *TPC* informa el suceso al *UI* y no solicita al *BE* la transferencia de *Ch*.

***UI* no entrega el *Ch* al *UR*.**
EL *UR* no podrá iniciar la *NIZKP* y, por tanto, la transferencia no puede iniciarse.

La verificación que hace el *TPC* del *NIZKP* iniciado por *UR* no es satisfactoria.
El *TPC* informa el suceso al *UR* y no solicita al *BE* la transferencia de *Ch*.

El *TPC* no es capaz de generar un *P* en base al *T* y al *CS* que el *UR* le proporcionó.
El *TPC* informa el suceso al *UI* y al *UR* y no solicita al *BE* la transferencia de *Ch*.

El *TPC* detecta que el *P* del *Ch* es distinto al valor generado por la función SHA-256 a partir de *T* y *CS*.
El *TPC* informa el suceso al *UI* y no solicita al *BE* la transferencia de *Ch*.

El *Ch* que aportó *UI* y el que aportó *UR* son distintos.
El *TPC* informa el suceso a *UI* y *UR* y no solicita al *BE* la transferencia de *Ch*.

El *T* del *UR* (*T'*) que aportaron *UI* y *UR* son distintos.
El *TPC* informa el suceso a *UI* y *UR* y no solicita al *BE* la transferencia de *Ch*.

La *Trx* no puede ser iniciada por el *BE*.
El *BE* informa el suceso al *TPC* para que a su vez informe a *UI* y a *UR*.

La *Trx* intenta consumir un *I* ya consumido.
El *N* informa el intento de doble gasto al *BE* para que este informa el suceso al *TPC* y este a su vez haga lo propio con *UI* y *UR*.

La *Trx* no pasa las restricciones impuestas por *SC.*
El *BE* informa el suceso al *TPC* para que a su vez informe a *UI* y a *UR*.

El *TPC* no recibe respuesta del *BE*.
El *TPC* está en disposición de preguntar al *BE* sobre el resultado de la transferencia presentando el *Ch* que se solicitó transferir.

***UI* o *UR* no reciben respuesta del *TPC*.**
UI y *UR* están en disposición de preguntar al *TPC* sobre el resultado de la transferencia presentando el *Ch* que se solicitó transferir.

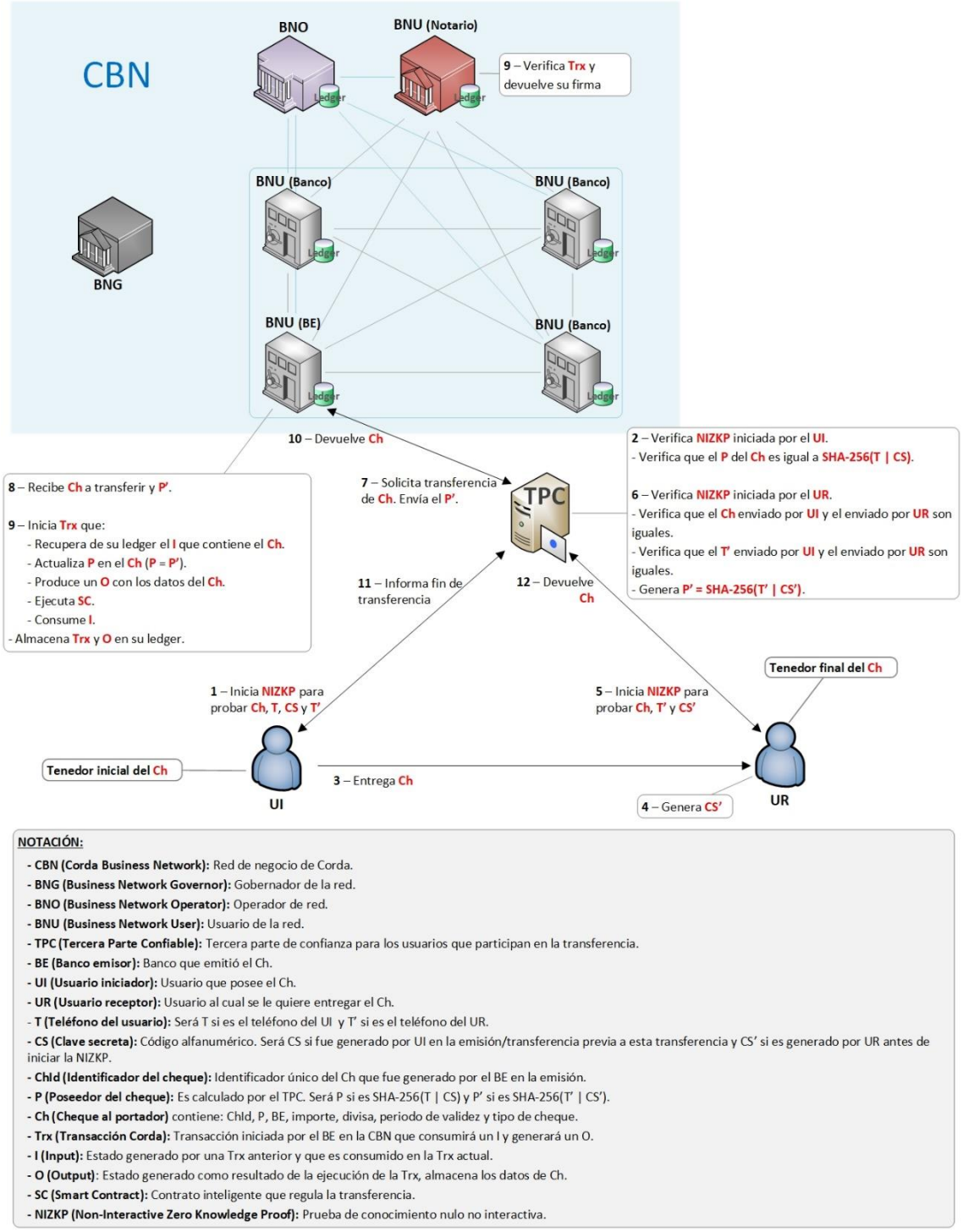


Figura 17. Protocolo de transferencia

3.8 Protocolo de depósito de un cheque

Este apartado tiene como objeto definir el protocolo a seguir cuando el propietario de un cheque solicita a su banco su depósito. La ejecución del protocolo conlleva un ingreso en la cuenta del cliente por valor del importe del cheque.

Las partes que intervienen en este protocolo son:

- Usuario propietario de un cheque que es cliente de un banco (*C*).
- Banco depositario (*Bde*).
- Banco emisor (*Bem*).
- Notario (*N*).

3.8.1 Pasos del protocolo

Al no definir el esquema el canal a través del cual *C* puede solicitar el depósito de un *Ch* a su banco*,* no se considera como paso del protocolo la autenticación de *C* ante su banco. Se parte de la certeza que *BD* ya ha identificado a su *C* y que este ya está en disposición de solicitar el depósito de un cheque.

Paso 1: *C*, como propietario del *Ch*, solicita al *BD* el depósito del *Ch* y le envía la *CS* que definió para el *Ch*.

Paso 2: *BD* recupera el *T* que internamente tiene asociado a *C*. Verifica que el *P* del *Ch* es igual al resultado obtenido del cálculo de la función SHA-256 sobre *T* y *CS*. Genera *P* en base a un *R* que *BD* define (se anota el *P* aquí calculado como *P'* para evitar confusiones con el *P* del *Ch* que se solicita depositar).

Paso 3: *BD* solicita a *BE* el depósito del *Ch* y le envía *P'*.

Paso 4: *BE* recibe *Ch* y *P'*.

Paso 5: *BE* invoca a su nodo Corda para iniciar una *Trx,* en la que se añade como participante a BD, que tenga como finalidad consumir el *I* con los datos de *Ch*. La ejecución de Trx supone que la CorDapp tiene que:

- Recuperar de su *ledger* el *I* que se corresponda con el *Ch* a depositar.
- Actualizar el *P* del *Ch* con el *P'* recibido.
- Hacer que el *SC* verifique tanto *I* como *O*.
- Marcar *I* como consumido.
- Almacenar en su *ledger O* y *Trx*.

La CorDapp de *N* tiene que validar la *Trx* y devolver su firma como comprobante de que *Trx* es válida para él y que *I* no había sido consumido previamente.

La CorDapp de *BD* tiene que validar la *Trx* y devolver su firma como comprobante de que *Trx* es válida para él. Además, debe almacenar en su *ledger* la *Trx*, I y *O*.

Paso 6: El *BE* devuelve el *Ch* al *BD*.

Paso 7: *BD* recibe el *Ch*.

Paso 8: *BD* invoca a su nodo Corda para iniciar una *Trx'* que tenga como finalidad consumir un *I,* que se corresponde con el *O* generado por la *Trx* en el paso 5*,* y no generar ningún *O*. La ejecución de *Trx'* supone que la CorDapp tiene que:

- Recuperar de su *ledger* el *I* que se corresponda con el *Ch* cuyo depósito se quiere hacer efectivo.
- Hacer que el SC verifique *I*.
- Marcar el *I* como consumido.
- Almacenar en su *ledger Trx'*.

La CorDapp de *N* tiene que validar la *Trx'* y devolver su firma como comprobante de que *Trx'* es válida para él y que *I* no había sido consumido previamente.

La CorDapp de *BE* tiene que validar la *Trx'* y devolver su firma como comprobante de que *Trx'* es válida para él. Además, debe almacenar en su *ledger* la *Trx'*.

Paso 9: *BD* confirma a *C* el depósito de *Ch* y el ingreso de su importe en su cuenta.

La figura 17 muestra de forma gráfica los distintos pasos de los que consta el protocolo y que han sido detallados en esta sección.

El depósito de un *Ch* sí supone un movimiento de fondos entre un *BE* y *BD*, aunque el protocolo no detalla cómo ha de hacerse este movimiento. Por el contrario, el protocolo si detalla como el *BD* del *Ch* tiene que hacer efectivo el depósito de dicho *Ch* en la cuenta de C.

Para el esquema el fin del depósito del *Ch* acaba con el paso 6 ya que a partir de este ningún otro banco podrá hacerse depositario de este *Ch* y, también, porque es posible implementar mecanismos que puedan volver a ejecutar los pasos siguientes sin necesidad de la intervención del *C*.

3.8.2 Incidencias o motivos de rechazo en el depósito

Este apartado hace constar los posibles motivos que durante el depósito de un *Ch* pudiesen surgir y provocasen que el proceso de depósito no se completase.

El *BD* no tiene ningún *T* asociado a *C*.

El *BD* no solicita a *BE* el depósito de *Ch*. *BD* solicita a *C* que informe un *T*.

El *BD* no es capaz de generar un *P* en base al *T* y al *CS* que *C* le proporcionó.

El *BD* no solicita a *BE* el depósito de *Ch*. BD informa a C el error.

El *BD* detecta que el *P* del *Ch* es distinto al valor generado por la función SHA-256 a partir de *T* y *CS*.

El *BD* informa el suceso al *C* y no solicita al *BE* el depósito de *Ch*.

El *BD* no es capaz de genera una *R* en base a la cual generar *P'*.

El *BD* informa el suceso al *C* y no solicita al *BE* el depósito de *Ch*.

El *BD* no puede contactar con el *BE*.

El *BD* informa el suceso al *C* para que intente el deposito más tarde.

La *Trx* no puede ser iniciada por el *BE*.

El *BE* informa el suceso al *BD* para que a su vez informe a su *C*.

La *Trx* intenta consumir un *I* ya consumido.

El *N* informa el intento de doble gasto al *BE* para que este informe el suceso al *BD* y este, a su vez, haga lo propio con su *C*.

La *Trx* no pasa las restricciones definidas en el *SC*.

El *BE* informa el suceso al *BD* para que este, a su vez, informe a su *C*.

El *BD* o el *N* no devuelven la firma que certifica la validez de la *Trx*.

El *BE* informa el suceso al *BD* para que este, a su vez, informe a su *C* que vuelva a intentarlo más tarde.

El *BD* no recibe respuesta del *BE*.

El *BD* está en disposición de preguntar al BE sobre el resultado del depósito presentando el *Ch* que se solicitó transferir.

La *Trx'* no puede ser iniciada por el *BD*.

El *BD,* una vez sabe que él es el depositario del *Ch* y que nadie más puede adquirir esa condición, intenta iniciar la *Trx'* más tarde.

La *Trx'* intenta consumir un *I* ya consumido.

El *N* informa el intento de doble gasto al *BD*. Esta situación supondría que el *BD* ya hizo efectivo el depósito del *Ch* en la cuenta de *A*.

La *Trx'* no pasa las restricciones definidas en el *SC*.

Esto supondría que el depósito no llega a hacerse efectivo nunca con lo que *C* no recibirá el importe del *Ch* en su cuenta. Esta casuística de lugar a una incidencia que habría que solucionar de manera manual.

El *N* no devuelve la firma que certifica la validez de la *Trx*.

Esto supondría que el depósito no llega a hacerse efectivo nunca con lo que *C* no recibirá el importe del *Ch* en su cuenta. Esta casuística de lugar a una incidencia que habría que solucionar de manera manual.

El *C* no recibe respuesta del *BD*.

C está en disposición de preguntar a su *BD* sobre el resultado del depósito presentando el *Ch* que se solicitó depositar.

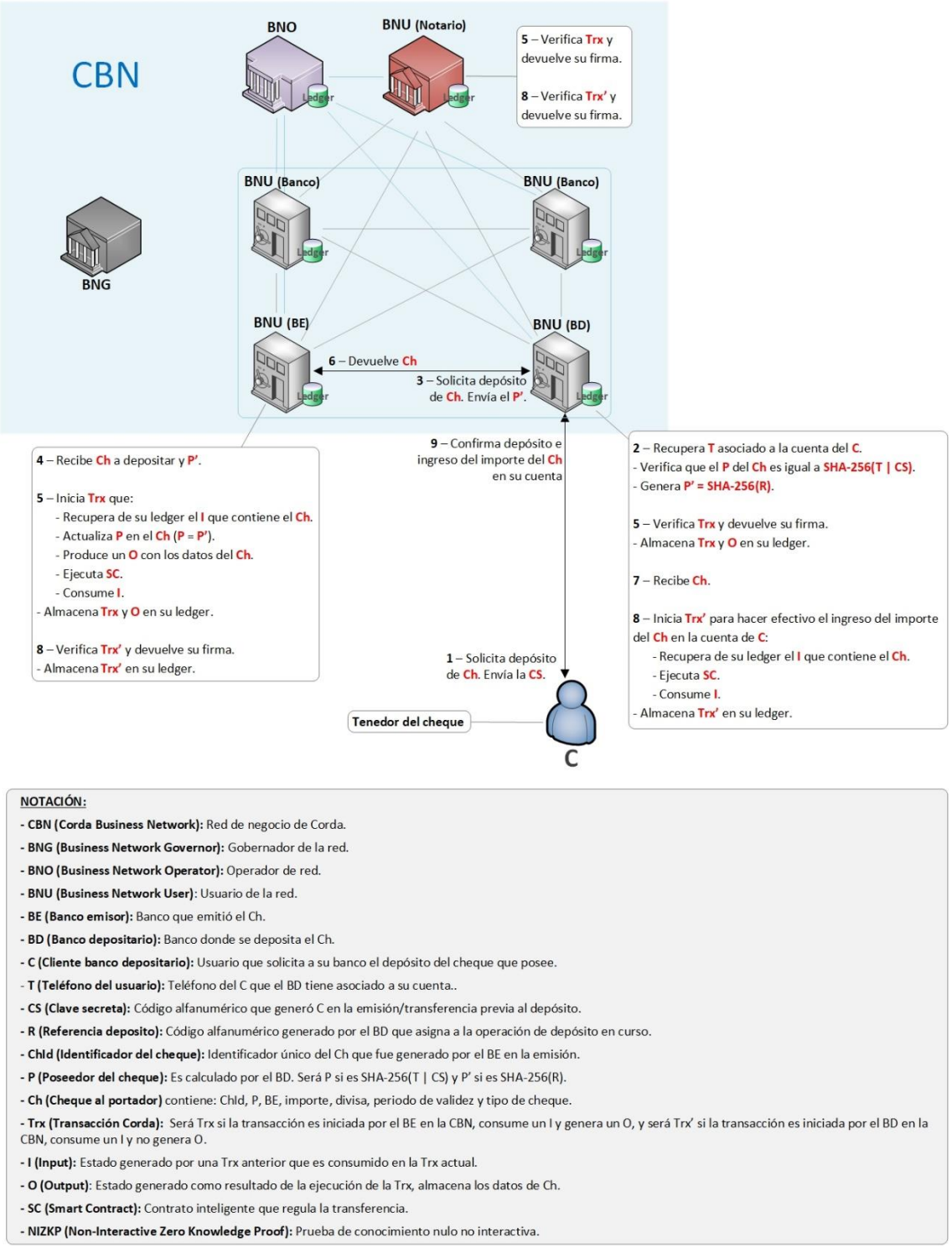


Figura 18. Protocolo de depósito

4. Prueba de concepto

Este capítulo tiene como objetivo el implementar una prueba de concepto (*PoC*) que pretende mostrar el funcionamiento del esquema anteriormente analizado y detallado haciendo uso de la plataforma Corda. Corda desde su inicio ha publicado varias versiones, siendo la versión 4 la última publicada.

Esta prueba pretende crear una aplicación Corda (CorDapp) que controle la emisión, transferencia y depósito de cheques y que garantice el cumplimiento de los requerimientos de seguridad definidos en el apartado 3.2 de este documento. La aplicación resultante, tal y como se indicó en el apartado 2.1.6, será un fichero con extensión .jar (Java ARchives) que será instalado y ejecutado en los nodos Corda que cada banco debe desplegar en un servidor dedicado.

El hecho de que la CorDapp permita conexiones controladas del exterior facilitará la integración del nodo Corda como un elemento más de los sistemas corporativos del banco. Así mismo, cualquier servicio ya existente en el banco podrá iniciar de manera remota transacciones Corda invocando al servicio de la CorDapp correspondiente.

El *PoC* que aquí se presenta ofrece 4 servicios. 3 de los cuales serán iniciados de forma remota por alguna aplicación o servicio del banco emisor de un cheque, estos servicios son: emisión, transferencia y depósito. Y el cuarto servicio, el cual se encarga de hacer efectivo el depósito del cheque en la cuenta de un cliente, será iniciado de forma remota por alguna aplicación o servicio del banco depositario del cheque. También, el *PoC* incluirá un cliente que sirva para iniciar de forma remota cada uno de los servicios arriba detallados.

Queda fuera de esta *PoC* el desarrollo de la prueba de conocimiento nula entre los usuarios que desean participar en la transferencia de un cheque y la Tercera Parte Confiable.

Es muy importante destacar que el despliegue de los nodos y la red generada como consecuencia de la ejecución de lo desarrollado en esta *PoC* se hace en modo desarrollo. Esto significa que en un entorno real no se aconseja bajo ninguna circunstancia la generación de la red ni de los nodos tal y como más abajo se define. Queda fuera del ámbito de este proyecto el definir como sería la instalación de nodos Corda en servidores reales, o virtuales, y el arranque y adhesión de esos nodos a una red.

4.1 Entorno de pruebas

Para esta *PoC* se ha usado un equipo con sistema operativo de *Windows® 10 Home* de 64 bits, con procesador *Intel ® Core ™ i7-7700HQ* y con 24 GB de memoria RAM. Para las pruebas se ha instalado la consola *MINGW64* (http://www.mingw.org/).

Como herramienta de desarrollo y lenguaje de programación se han usado *Eclispe Oxygen 3a* y *Java* a pesar de que Corda orienta toda su documentación para el uso de *IntelliJ* y *Kotlin.*

Como el objetivo es crear una CorDapp (fichero .jar) es necesario instalar una máquina virtual de *Java (JVM)*. Corda, en su versión 4, requiere al menos una versión *Java* igual o superior a la 1.8.0_171 pero inferior a *Java* 9 y posteriores. En este caso para el desarrollo se ha instalado la *JDK 1.8.0_202* que ofrece *Oracle®* en su sitio web (https://www.oracle.com/technetwork/java/index.html)

Para facilitar los desarrollos Corda tiene un repositorio público en *GitHub* (https://github.com/corda), de donde poder descargar plantillas y proyectos de ejemplo. También, ofrece una API (https://docs.corda.net/api/javadoc/index.html) donde consultar todas las clases de su librería *net.corda*. Para el inicio del desarrollo, y dado que se va a codificar en Java, se ha descargado la siguiente plantilla: https://github.com/corda/cordapp-template-java.

Una vez descargada la plantilla se hace necesario instalar *Gradle* en el equipo. *Gradle* es una herramienta open-source que permite, entre otras cosas, la automatización de compilaciones de código, la gestión de dependencias, la generación de ficheros .jar, .war y .ear, la ejecución de plugins, etc. En los siguientes enlaces se puede obtener más información sobre las características de *Gradle* y su instalación:

- https://gradle.org/features/
- https://gradle.org/install/

En el fichero de configuración de *Gradle* (*build.gradle*) de la plantilla descargada vienen predefinidas una serie de tareas que permiten la ejecución y depuración de las CorDapp durante la fase de desarrollo. Una de estas tareas permite el despliegue de nodos en el directorio de trabajo, cargando en ellos la CorDapp que se está desarrollando, y arrancar los nodos desplegados con el objetivo de testear la aplicación. En secciones posteriores se hará referencia nuevamente al fichero *build.gradle* y a la configuración de nodos que se ha hecho para esta PoC.

Como se comentó en la sección 2.1 cada nodo Corda tendrá su propia ledger. Los nodos que se desplieguen haciendo uso de la tarea que se introdujo en el párrafo anterior crearán y harán uso de una base de datos *H2* (https://www.h2database.com). Para poder consultar el esquema de la ledger y los datos que se almacenan por cada

ejecución de una transacción Corda finalizada correctamente se necesita instalar un cliente de *H2*.

4.2 Estructura y análisis de la prueba de concepto

Esta prueba de concepto consta de dos proyectos *TFM-PoC_CordaNode* y *TFM-PoC_RPCClients.* El primero de ellos tiene como finalidad el generar los ficheros .jar que se instalarían en los nodos Corda de bancos y de notarios, y el segundo contiene clientes de esos nodos Corda.

Como se ha indicado anteriormente en diversas ocasiones un CorDapp no es más que un fichero con extensión .jar que puede ser ejecutado en una JVM y que contiene archivos con extensión .class que se corresponden con clases compiladas que fueron codificadas en lenguaje Java o Kotlin. Haciendo un análisis de los tipos de clases, en cuanto a contenido y a funcionalidad, que se pueden encontrar en una aplicación CorDapp conviene destacar tres tipos principalmente:

- **Estados:** Son aquellas clases que definen objetos en su interior y que se usarán como parámetros de entrada y de salida en una transacción. Son parte de la transacción y como tal se almacenan en la ledger.

  Para que una clase sea considerada un estado, debe implementar la clase *net.corda.core.contracts.ContractState*, o alguna de las que heredan esta, e implementar el método *getParticipants()*. Este definirá los nodos que participarán en la transacción y que, por tanto, deben aceptarla.

  Los estados son piezas de información que pueden viajar a través de la red Corda, en este caso habría que anotar la clase como *@CordaSerializable*. También es necesario relacionar la clase con un contrato, para habría que anotarla como *@BelongsToContract(Contract.class)*, donde *Contract.class* será la clase que implementa el contrato.

- **Contratos:** Serían los *smart contract* de la aplicación. Son clases que, codificadas en base a un acuerdo de funcionamiento de un sistema, o en base a un acuerdo entre las partes que intervienen en una transacción, imponen ciertas restricciones durante la ejecución de la transacción.

  Un nodo da como válida una transacción cuando sus estados de entrada y de salida han sido validados satisfactoriamente. Se considera que una transacción ha finalizado de forma correcta cuando ha sido validada y aceptada por todos los nodos que participan en la transacción.

  Toda clase que defina un contrato debe implementar la clase *net.corda.core.contracts.Contract* y su método *verify()*. Será este método quien imponga las restricciones sobre la transacción en base a su propósito y, quien

en caso de verificar que alguna restricción no se cumple, lanzará una excepción abortando la ejecución de la misma.

- **Flujos:** Son aquellas clases que ofrecen un servicio que puede ser invocado de forma remota usando RPC. Su instanciación puede iniciar la ejecución de una transacción Corda.

  Una clase será considerada un flujo cuando sea anotada con *@InitiatingFlow* y cuando herede la clase *net.corda.core.flows.FlowLogic*. Esta herencia obliga a implementar la función *call()* que se encargará de:

  - Iniciar la transacción y asignarle un notario de la red.
  - Definir las partes que deben firmar la transacción como prueba de que la aceptan.
  - Añadir a la transacción los estados de entrada a consumir y los de salida a generar.
  - Adjuntar a la transacción documentos en caso de estimarse oportuno.
  - Invocar al contrato correspondiente para que valide los estados de la transacción.
  - Enviar la transacción al resto de participantes para que la validen.
  - Enviar la transacción al notario para que compruebe que no intenta consumir un estado consumido previamente.
  - Almacenar transacción en la ledger del nodo.

  Un flujo también puede ser invocado por un cliente RPC si dicha clase es anotada con *@StartableByRPC*. El constructor de la clase define los parámetros que un cliente RPC debe enviar al flujo cuando en la llamada.

  También es posible definir flujos en los otros nodos que participan en una transacción y que no son el iniciador. Estas clases deben ser anotadas como *@InitiatedBy(Flow.class)* donde *Flow.class* será la clase con la notación *@InitiatingFlow* a la que queremos vincular este flujo. Esta clase también debe heredar de la clase *net.corda.core.flows.FlowLogic* e implementar su método *call()* y, también puede llevar a cabo verificaciones sobre la transacción en ejecución.

En cuanto a los clientes RPC, serán clases capaces de invocar a flujos de un nodo Corda haciendo uso de las siguientes clases:

- net.corda.core.utilities.NetworkHostAndPort
- net.corda.client.rpc.CordaRPCClient
- net.corda.client.rpc.CordaRPCConnection n
- net.corda.core.messaging.CordaRPCOps

En todo proyecto que se codifiquen clientes RPC será necesario que se incluyan las clases de estados. Estas permitirán hacer una correcta serialización de los objetos que puede contener la transacción.

4.2.1 TFM-Poc_CordaNode

Como ya se ha indicado, este proyecto ha sido desarrollado partiendo de una de las plantillas que ofrece Corda en *GitHub*. Por la configuración de dicha plantilla, *TFM-PoC_CordaNode* será un multi-proyecto que inicialmente contiene los proyectos: *clients, contracts* y *workflows*. Dado que el objetivo de este proyecto es crear una CorDapp que sea ejecutada en los nodos se elimina el proyecto *clients* para que no esté contenido en la compilación final. La figura 19 muestra la estructura del proyecto incluyendo: las clases que lo componen, los archivos *build.gradle* de cada uno y los scripts generados para facilitar el arranque de los nodos y la automatización de tareas.

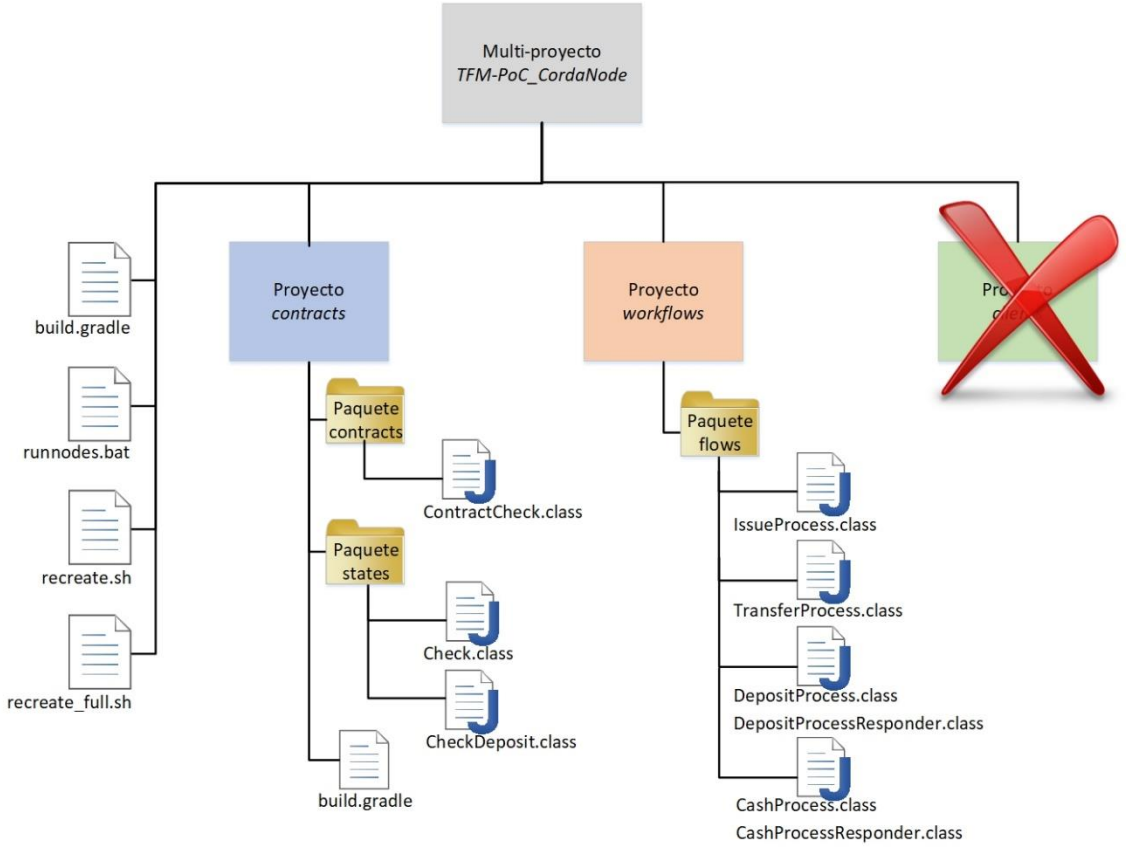


Figura 19. Estructura proyecto TFM-PoC_CordaNode

La plantilla usada define tantos ficheros *build.gradle* como proyectos forman *TFM-PoC_CordaNode*. Esto significa que habrá uno en el directorio raíz del multi-proyecto y otro en cada directorio de los proyectos *contracts* y *workflows*. *build.gradle* permite definir configuraciones de compilación, dependencias del proyecto, tareas, etc.

A continuación, se enumeran las clases que se han definido en los proyectos *contracts* y *workflows* agrupadas por tipo. El proyecto *contracts* contiene las clases de tipo estado y contrato mientras que el proyecto *workflows* contiene las de tipo flujo.

- **Clases de tipo estado:**
  - ***Check*:** Clase que almacena los datos del cheque y el banco emisor del mismo.

    Participa en las transacciones que se ejecutan durante los procesos de emisión, transferencia y depósito. El proceso de emisión creará un estado *Check*, el proceso de transferencia consume y crea un objeto *Check* y el proceso de depósito consume un proceso *Check*.
  - ***CheckDeposit:*** Clase que almacena los datos de *Ch* y el banco emisor y depositario del mismo.

    Participa en las dos transacciones que se llevan a cabo durante el proceso de depósito:
    - Será estado generado en la transacción que es iniciada por el banco emisor a petición del banco depositario.
    - Será estado consumido en la transacción que hace efectivo el depósito del cheque haciendo el ingreso en la cuenta del cliente.
- **Clases de tipo contrato:**
  - ***ContractCheck:*** Clase que define las restricciones que deben cumplir los estados de los que hace uso la transacción en cada uno de los flujos. Define restricciones diferentes por cada tipo de flujo que existe en el proyecto.

    A modo de prueba para el flujo de emisión de un cheque se ha definido que la moneda del cheque solo puede ser euro (EUR) o dólar (USD) y que en ningún caso el importe será superior a 1.000€ o 1.000$.
- **Clases de tipo flujo:**
  - **IssueProcess**: La invoca el banco emisor del cheque y como consecuencia inicia una transacción Corda que no consume ningún estado, no tiene entradas, y genera un estado de salida (*Check*).

    El flujo declara un constructor para inicializar el banco emisor del cheque y los siguientes campos de este: Importe, moneda, periodo de validez, tipo y poseedor. El campo poseedor le debe ser informado como una cadena de caracteres hexadecimales que son el resultado de aplicar una función SHA-256 sobre el teléfono y la clave secreta del usuario que solicita la emisión del cheque.

    La transacción es firmada por el banco emisor y validada por el notario. Solo el banco emisor almacena la transacción en su ledger.

- **TransferProcess**: La invoca el banco emisor del cheque y como consecuencia inicia una transacción Corda que consume un estado *Check* y genera otro estado *Check* de salida.

  El flujo declara un constructor para inicializar el banco emisor, el identificador único del cheque y dos poseedores, uno que se corresponde con el tenedor actual del cheque y otro que se corresponde con el usuario al que se le quiere entregar.

  La transacción es firmada por el banco emisor y el notario. El notario, además de validar la transacción, verifica que no consume un estado *Check* ya consumido. El banco emisor almacena la transacción actual en su ledger y el notario hará lo propio con la transacción que origino el estado *Check* que aquí se consumió.

- **DepositProcess**: La invoca el banco emisor del cheque, a petición del que será el banco depositario, y como consecuencia inicia una transacción Corda que consume un estado *Check* y genera otro estado *CheckDeposit* de salida

  El flujo declara un constructor para inicializar el banco emisor, el banco depositario, el identificador único del cheque y dos poseedores, uno que se corresponde con el usuario que solicita el ingreso del cheque en su cuenta y otro que se corresponde con una referencia que asigna el banco depositario al proceso de depósito.

  La transacción es firmada por el banco emisor, el banco depositario y el notario. El notario, además de validar la transacción, verifica que no consume un estado *Check* ya consumido. El banco emisor almacena la transacción actual en su ledger, el notario hará lo propio con la transacción que origino el estado *Check* que aquí se consumió y el banco depositario almacenará la cadena completa de transacciones que dieron origen a esta.

- **DepositProcessResponder:** La inicia el banco depositario del cheque a raíz de la inicialización del flujo *DepositProcess* por el banco emisor. Hace comprobaciones sobre los estados que consume.

- **CashProcess:** La inicia el banco depositario del cheque y como consecuencia inicia una transacción Corda que consume un estado *CheckDeposit* y no genera ningún estado de salida.

  El flujo declara un constructor para inicializar el banco depositario, el identificador único del cheque y el poseedor del mismo, este se corresponde con el resultado que generó la función SHA-256 sobre la referencia que asignó el banco depositario al proceso de depósito.

La transacción es firmada por el banco emisor, el banco depositario y el notario. El notario, además de validar la transacción, verifica que no consume un estado *CheckDeposit* ya consumido. El banco emisor y el banco depositario almacenan la transacción actual en su ledger y el notario hará lo propio con la transacción que origino el estado *CheckDeposit* que aquí se consumió.

Con este flujo se da por cerrado el ciclo de vida del cheque al no generarse ningún estado de salida y al haberse ya consumido todos los estados anteriores.

- **CashProcessResponder:** La inicia el banco emisor del cheque a raíz de la inicialización del flujo *CashProcess* por el banco depositario. Hace comprobaciones sobre el estado que se consume.

Como se puede ver en la figura 19, y ya se comentó anteriormente, el multi-proyecto contiene un archivo *build.gradle* el cual declara la tarea *deployNodes*. Esta tarea permite definir nodos Corda que serán desplegados en el espacio de trabajo al ejecutar el siguiente comando: *gradle clean deployNodes*. También, dentro del directorio raíz del multi-proyecto, se incluye el script *runnodes.bat* que al ser ejecutado arranca todos los nodos que la tarea *deployNodes* generó. La forma de definir la configuración de un nodo en la tarea *deployNodes* es la que muestra la figura 20.

```
node {
    name "O=Bank1,L=Philadelpia,C=US"

    p2pAddress "localhost:11011"

    rpcSettings {
        address("localhost:11012")
        adminAddress("localhost:11013")
    }
    rpcUsers = [[ user: "bank1", "password": "bank1*!", "permissions": ["ALL"]]]

    h2Port 11014

    extraConfig = [
        "dataSourceProperties.dataSource.user" : "bank1",
        "dataSourceProperties.dataSource.password" : "bank1*!"
    ]
}
```

Figura 20. Configuración nodo Corda en la tarea deployNodes de build.gradle

Los parámetros de los que consta la configuración del nodo son:

- **name**: Identificador único del nodo. Tendrá formato *DN* (Distinguished Name) y estará formado por el Organization (*O*), Locality (*L*) y Country (*C*). Se corresponderá con el campo *Subject* del certificado X.509 que la tarea *deployNodes* genera para cada nodo y almacena en su correspondiente carpeta *certificates*.
- **p2pAddress**: Dirección y puerto a través del cual el nodo establecerá conexiones con otros nodos de la red.

- **rpcSettings**: Dirección y puerto a los cuales los clientes RPC deben hacer sus llamadas (**address**) o dirección y puerto que servirá como punto de acceso remoto para administrar el nodo (**adminAddress**).
- **rpcUser**: Define los usuarios (**user**) y las contraseñas (**password**) que deben usar los clientes RPC para invocar de manera remota los flujos que define el nodo. Así mismo, por cada usuario, se permiten definir los flujos a los que tendrán acceso los clientes RPC (**permissions**), en el ejemplo de la figura todos los flujos que define el nodo serán accesibles.
- **h2port**: Puerto que habilita el nodo para interactuar con su ledger, base de datos H2.
- **dataSourceProperties.dataSource.user** y **.password:** Usuario y contraseña que permite conectarse a la base de datos .

En el caso de que el nodo a configurar sea un notario se recomienda aplicar una política restrictiva en cuanto al número de usuarios y de permisos que se le otorgan a cada usuario. Por otro lado, en la configuración de un notario podemos añadir el elmento **notary = [validating : true]** que indica que el notario participa en el consenso de validez de la transacción.

La figura 21 muestra la configuración de los nodos Notary y Bank2 que se han definido en la PoC. La configuración del Bank1 se corresponde con la de la figura 20.

```
node {
    name "O=Notary,L=New York,C=US"

    notary = [validating : true]

    p2pAddress "localhost:11001"

    rpcSettings {
        address("localhost:11002")
        adminAddress("localhost:11003")
    }

    h2Port 11004

    extraConfig = [
        "dataSourceProperties.dataSource.user" : "notary",
        "dataSourceProperties.dataSource.password" : "notary*!"
    ]
}

node {
    name "O=Bank2,L=Philadelpia,C=US"

    p2pAddress "localhost:11021"

    rpcSettings {
        address("localhost:11022")
        adminAddress("localhost:11023")
    }
    rpcUsers = [[ user: "bank2", "password": "bank2*!", "permissions": ["ALL"]]]

    h2Port 11024

    extraConfig = [
        "dataSourceProperties.dataSource.user" : "bank2",
        "dataSourceProperties.dataSource.password" : "bank2*!"
    ]
}
```

Figura 21. Configuración de los nodos Corda Notary y Bank2 usados en la PoC

El siguiente enlace ofrece una lista completa de todos los campos que se pueden incluir en la configuración de un nodo. https://docs.corda.net/corda-configuration-file.html#configuration-file-fields.

En cuanto a la compilación del proyecto, despliegue de nodos y arranque de estos, se ha añadido en el directorio raíz del multi-proyecto un script, llamado *recreate_full.sh*, cuyo contenido se muestra en la figura 22, y que ejecuta las siguientes sentencias:

- ***gradle clean build -x test***: Compila los proyectos *contracts* y *workflows* sin ejecutar los tests del proyecto.

  Dentro de la configuración del *build.gradle* del multiproyecto se ha definido que se genere un fichero .jar por cada uno de los proyectos (*contracts* y *workflows)*. Se podría modificar esta configuración para que se generase un único .jar que aunase ambos proyectos.

- ***gradle clean deploynodes -x test***: Ejecuta la tarea *deployNodes* definida en *build.gradle* sin ejecutar los tests del proyecto.

  El despliegue de nodos conlleva la creación de un directorio por nodo dentro de la carpeta *build/nodes* del espacio de trabajo. A su vez, cada directorio de nodo contendrá el fichero de configuración del nodo, una carpeta con los certificados que permiten establecer la conexión TLS y firmar las transacciones, y un directorio con los ficheros .jar generados con el comando anterior.

- ***build/nodes/runnodes***: Arranca cada uno de los nodos desplegados en el punto anterior en una JVM que permitirá ejecutar transacciones Corda invocando los flujos correspondientes.

  En este punto, la red Corda, y sus nodos, sobre la que se van a ejecutar las pruebas ya estaría disponible.

```
#!/bin/bash

gradle clean build -x test
echo -e "\nProject compiled\n"

gradle clean deploynodes -x test
echo -e "\nNodes deployed\n"

build/nodes/runnodes
echo -e "\nNodes running\n"
```

Figura 22. Script recreate_full.sh

Para esta PoC el script *recreate_full.sh* ha sido ejecutado en la consola MINGW64, conviene destacar la estructura *NetworkParameters* que se informa por consola como consecuencia de la ejecución del comando *gradle clean deploynodes -x test*. La figura 23 muestra el formato y el contenido de esta estructura. Corda publica en la web https://docs.corda.net/network-map.html#network-parameters más información sobre los atributos que esta estructura puede contener.

```
> Task :deployNodes
Running Cordform task
Deleting D:\Oscar\MISTIC\TFM\Workspace\TFM-PoC_CordaNode\build\nodes
Bootstrapping local test network in D:\Oscar\MISTIC\TFM\Workspace\TFM-PoC_CordaNode\build\nodes
Generating node directory for Bank1
Generating node directory for Bank2
Generating node directory for Notary
Waiting for all nodes to generate their node-info files...
Distributing all node-info files to all nodes
Loading existing network parameters... none found
Gathering notary identities
Generating contract implementations whitelist
New NetworkParameters {
      minimumPlatformVersion=4
      notaries=[NotaryInfo(identity=O=Notary, L=New York, C=US, validating=true)]
      maxMessageSize=10485760
      maxTransactionSize=524288000
      whitelistedContractImplementations {

      }
      eventHorizon=PT720H
      packageOwnership {

      }
      modifiedTime=2019-06-11T08:48:29.256Z
      epoch=1
  }
Bootstrapping complete!
```

Figura 23. Estructura NetworkParameters

Como datos más importantes de la estructura *NetworkParameters* conviene destacar los elementos *minimumPlatformVersion* y *notaries*. *minimumPlatformVersion* identifica la versión mínima de las CorDapps que los nodos pueden ejecutar en esta red y el elemento *notaries* define la identidad de los notarios adheridos la red.

Como resultado final de la ejecución del script este arrancará una JVM, tal y como la que se muestra en la figura 24, por nodo. Haciendo un análisis de la información que se muestra al arrancar la máquina podemos comprobar que las direcciones y puertos informados se corresponden con los que se configuraron en la tarea *deployNodes* del archivo *build.gradle*. Además, se muestra la versión del nodo Corda arrancado, el nombre del mismo y se informa que el nodo ha sido arrancado en modo desarrollo.

```
C:\Program Files\Java\jdk1.8.0_202\jre\bin\java.exe
Listening for transport dt_socket at address: 5006

←[91m   ______                  __
  / ____/     _________/ /___._
 / /      __  / ___/ __  / __ `/         ←[94mThere was an earthquake in California,
←[91m/ /___  /_/ / /  / /_/ / /_/ /          ←[94ma local bank went into de-fault.
←[91m\____/     /_/   \__._/\__._/←[m

←[99;1m--- Corda Open Source 4.0 (503a2ff) ----------------------------------------------------------

←[m
Logs can be found in                    : D:\Oscar\MISTIC\TFM\Workspace\TFM-PoC_CordaNode\build\nodes\Bank2\logs
←[31m! ATTENTION: This node is running in development mode!  This is not safe for production deployment.←[0m
Database connection url is              : jdbc:h2:tcp://localhost:11024/node
Advertised P2P messaging addresses      : localhost:11021
RPC connection address                  : localhost:11022
RPC admin connection address            : localhost:11023
Jolokia: Agent started with URL http://127.0.0.1:7006/jolokia/
Loaded 2 CorDapp(s)                     : Contract CorDapp: Template CorDapp version 1 by vendor Corda Open Source with
licence Apache License, Version 2.0, Workflow CorDapp: Template Flows version 1 by vendor Corda Open Source with licence
 Apache License, Version 2.0
Node for "Bank2" started up and registered in 27.44 sec


Welcome to the Corda interactive shell.
Useful commands include 'help' to see what is available, and 'bye' to shut down the node.
```

Figura 24. Ejecución de nodo Corda en JVM

Llegados a este punto el sistema, con sus nodos ejecutándose, ya está en disposición de emitir, transferir y depositar cheques al portador.

4.2.2 TFM_PoC_RPCClients

Tiene como propósito el crear clientes que a través de llamadas RPC sean capaces de iniciar los flujos que los nodos Corda desplegados en el proyecto *TFM_PoC_CordaNodes* ofrecen. A continuación se definen las clases que se han implementado en el proyecto. Cada una de estas clases contiene un método *callFlow()* que inicia la conexión con los nodos Corda e invoca de forma remota a cada uno de los flujos definidos en la CorDapp:

- ***IssueCheckClient***: Define un cliente RPC que permite solicitar a un banco la emisión de un cheque al portador invocando al flujo *IssueProcess*. En la llamada al flujo informa los datos del cheque: banco emisor, importe, moneda, periodo de validez, tipo y poseedor. El poseedor del cheque será informado como un String cuyo contenido es el resultado en hexadecimal de aplicar la función SHA-256 sobre el teléfono y clave secreta que define el cliente que solicita el cheque.

  Si la transacción finaliza correctamente imprime por consola un objeto *Check* y el identificador de la transacción Corda (*TrxId*).

  Esta clase sería invocada por los sistemas internos del banco emisor que reciben de un cliente la solicitud de emisión de un cheque.

- ***TransferCheckClient***: Define un cliente RPC que permite solicitar al banco emisor de un cheque la transferencia de dicho cheque invocando al flujo *TransferProcess*. En la llamada al flujo informa el banco emisor, el identificador único del cheque y los poseedores actual y final del cheque. Ambos poseedores son informados como un String cuyo contenido es el resultado en hexadecimal de aplicar la función SHA-256 sobre un teléfono y clave secreta.

  Si la transacción finaliza correctamente imprime en consola un objeto *Check* y el identificador de la transacción Corda (*TrxId*).

  Esta clase sería invocada por los sistemas internos del banco emisor que reciben de la Tercera Parte Confiable la solicitud de transferir un cheque.

- ***DepositCheckClient***: Define un cliente RPC que permite solicitar al banco emisor de un cheque el depósito de dicho cheque invocando al flujo *DepositProcess*. En la llamada al flujo informa el banco emisor, el banco depositario, el identificador único del cheque y los poseedores actual y final del cheque. Ambos poseedores son informados como un String cuyo contenido es el resultado en hexadecimal de aplicar la función SHA-256 sobre un teléfono y clave secreta o sobre una referencia que el banco depositario asigna al proceso de depósito.

  Si la transacción finaliza correctamente imprime en consola un objeto *CheckDeposit* y el identificador de la transacción Corda (*TrxId*).

  Esta clase sería invocada por los sistemas internos del banco emisor que reciben del banco depositario la solicitud de depositar un cheque.

- ***CashCheckClient***: Define un cliente RPC que permite solicitar al banco depositario de un cheque hacer efectivo el depósito del mismo en la cuenta del cliente invocando al flujo *CashProcess*. En la llamada al flujo informa el banco el banco depositario, el identificador único del cheque y el poseedor actual del cheque. Este poseedor es informado como un String cuyo contenido es el resultado en hexadecimal de aplicar la función SHA-256 sobre la referencia que el banco depositario asignó en el proceso de depósito.

  Si la transacción finaliza correctamente imprime en consola el identificador de la transacción Corda (*TrxId*).

  Esta clase sería invocada por los sistemas internos del banco depositario como consecuencia de la necesidad de ingresar el dinero en la cuenta del cliente.

4.3 Resultados tests PoC

Esta sección pretende recoger las evidencias y los resultados obtenidos de la ejecución de los tests efectuados sobre la PoC desarrollada. A continuación, se definen los tests ejecutados y el objetivo que se persigue con cada uno de ellos.

- **Test 1**: Emisión de un cheque.
- **Test 2**: Transferencia de un cheque.
- **Test 3**: Depósito de un cheque.
- **Test 4**: Hacer efectivo el depósito de un cheque en la cuenta del cliente.
- **Test 5**: Transferir varias veces el mismo cheque de forma consecutiva (Transferibilidad)
- **Test 6**: Evitar la transferencia de un cheque que ya ha sido entregado a otro (Detección doble uso).
- **Test 7**: Detección del intento de un banco de emitir un cheque en nombre de otro.
- **Test 8**: Atomicidad en la transferencia de un cheque.
- **Test 9**: Comprobar que el anonimato de las partes que intervienen en la transferencia está asegurado.

Por cada test realizado se informan:

- Los datos de entrada.
- Los resultados de la ejecución del test que se pintan por consola: identificador de la transacción ejecutada y datos del cheque generado, si la transacción lo genera.
- Las evidencias del registro de las transacciones que son almacenadas en la ledger de cada nodo. Estas evidencias solo se incluirán cuando sea relevante para la demostración.

- **Test 1:**

Objetivo: Emisión de un cheque al portador desde Bank1.

Datos de entrada:

```
- Nodo Corda desde el que emitir el cheque (Bank1 (1) or Bank2 (2)): 1
- Banco emisor del cheque Bank1 (1) or Bank2 (2): 1
- Importe (Número entero): 100
- Moneda (ISO 4217): eur
- Fecha caducidad (Formato: yyyy-MM-dd): 2019-12-12
- Teléfono del cliente: 630519103
- Clave secreta del cliente: oscar
       Poseedor del cheque (Resultado SHA-256):
       0F42DADE374186C6236F7334D230E401BCD3BF0ECC425AA7750E3F6D933DACC9
```

Resultado de la ejecución:

```
Cheque emitido en la transacción con Id:
918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE
Datos del cheque (Estado com.template.states.Check):
       - Banco emisor: O=Bank1, L=Philadelpia, C=US
       - Importe: 100
       - Moneda: EUR
       - Fecha de inicio: 2019-06-11T11:11:41.213Z
       - Fecha de caducidad: 2019-12-12T23:59:59.990Z
       - Tipo de cheque: BEARER
       - Poseedor: 0F42DADE374186C6236F7334D230E401BCD3BF0ECC425AA7750E3F6D933DACC9
       - Id Único: e542ba4d-a525-493c-8ec1-e0e31af33df1
```

Evidencias ledger: En base al resultado de la ejecución, únicamente la ledger de Bank1 debe almacenar la transacción con Id *918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE* que generó un estado de salida de tipo *com.template.states.Check*.

Ledger Bank1:

```
SELECT VLS.UUID, VS.TRANSACTION_ID, VS.CONSUMED_TIMESTAMP, CONTRACT_STATE_CLASS_NAME
FROM VAULT_STATES AS VS, VAULT_LINEAR_STATES AS VLS
WHERE  VLS.UUID = 'e542ba4d-a525-493c-8ec1-e0e31af33df1' AND VLS.TRANSACTION_ID = VS.TRANSACTION_ID
ORDER BY VS.RECORDED_TIMESTAMP;
```

UUID	TRANSACTION_ID	CONSUMED_TIMESTAMP	CONTRACT_STATE_CLASS_NAME
e542ba4d-a525-493c-8ec1-e0e31af33df1	918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE	*null*	com.template.states.Check

(1 fila, 2 ms)

- **Test 2:**

Objetivo: Transferencia del cheque con Id *e542ba4d-a525-493c-8ec1-e0e31af33df1* iniciada desde Bank1.

Datos de entrada:

```
- Nodo Corda desde el que transferir el cheque (Bank1 (1) or Bank2 (2)): 1
- Banco emisor del cheque Bank1 (1) or Bank2 (2): 1
- Teléfono del poseedor actual del cheque: 630519103
- Clave secreta del poseedor actual del cheque: oscar
       Poseedor actual del cheque (Resultado SHA-256):
       0F42DADE374186C6236F7334D230E401BCD3BF0ECC425AA7750E3F6D933DACC9
- Teléfono del usuario al que se le quiere entregar el cheque: 630630630
- Secret key del usuario al que se le quiere entregar el cheque: pepe
       Futuro poseedor del cheque (Resultado SHA-256):
       77BD964E201D01C42C330BCF8A3F45C2C0B955DA518C04B73A239627C16328B7
- Id único del cheque: e542ba4d-a525-493c-8ec1-e0e31af33df1
```

Resultado de la ejecución:

```
Se ha transferido el cheque en la transacción con Id:
E7A7E96F611AA075EE60FCB6924301469C21547496B59ECC186B1289820A8CE0
Datos del cheque (Estado com.template.states.Check):
       - Banco emisor: O=Bank1, L=Philadelpia, C=US
       - Importe: 100
       - Moneda: EUR
       - Fecha de inicio: 2019-06-11T11:11:41.213Z
       - Fecha de caducidad: 2019-12-12T23:59:59.990Z
       - Tipo de cheque: BEARER
       - Poseedor: 77BD964E201D01C42C330BCF8A3F45C2C0B955DA518C04B73A239627C16328B7
       - Id Único: e542ba4d-a525-493c-8ec1-e0e31af33df1
```

Evidencias ledger: En base al resultado de la ejecución, la ledger de Bank1 debe almacenar la nueva transacción (*E7A7E96F611AA075EE60FCB6924301469…*) y marcar como consumido el estado (*com.template.states.Check*) generado en la emisión. Así mismo, la ledger del Notary almacenará la transacción *918D43E086821CB482DC9AF8DA5…* cuyo estado fue consumido por la transferencia.

Ledger Bank1:

```
SELECT VLS.UUID, VS.TRANSACTION_ID, VS.CONSUMED_TIMESTAMP, CONTRACT_STATE_CLASS_NAME
FROM VAULT_STATES AS VS, VAULT_LINEAR_STATES AS VLS
WHERE  VLS.UUID = 'e542ba4d-a525-493c-8ec1-e0e31af33df1' AND VLS.TRANSACTION_ID = VS.TRANSACTION_ID
ORDER BY VS.RECORDED_TIMESTAMP;
```

UUID	TRANSACTION_ID	CONSUMED_TIMESTAMP	CONTRACT_STATE_CLASS_NAME
e542ba4d-a525-493c-8ec1-e0e31af33df1	918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE	2019-06-11 13:30:21.618	com.template.states.Check
e542ba4d-a525-493c-8ec1-e0e31af33df1	E7A7E96F611AA075EE60FCB6924301469C21547496B59ECC186B1289820A8CE0	*null*	com.template.states.Check

(2 filas, 2 ms)

Ledger Notary:

```
select * from NODE_TRANSACTIONS where (tx_id = '918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE' or
tx_id = 'E7A7E96F611AA075EE60FCB6924301469C21547496B59ECC186B1289820A8CE0');
```

TX_ID	TRANSACTION_VALUE
918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE	636f7264610100000080c562000000000001d000002a6d0000000300a3226e657

(1 fila, 8 ms)

- **Test 3:**

Objetivo: Depósito con Id *e542ba4d-a525-493c-8ec1-e0e31af33df1* iniciado desde Bank1.

Datos de entrada:

```
- Nodo Corda desde el que depositar el cheque (Bank1 (1) or Bank2 (2)): 1
- Banco emisor del cheque Bank1 (1) or Bank2 (2): 1
- Banco depositario del cheque Bank1 (1) or Bank2 (2): 2
- Teléfono del poseedor del cheque: 630630630
- Clave secreta del poseedor actual del cheque: pepe
       Poseedor actual del cheque (Resultado SHA-256):
       77BD964E201D01C42C330BCF8A3F45C2C0B955DA518C04B73A239627C16328B7
- Referencia de operación de depósito: 1
       Poseedor en base a la referencia (Resultado SHA-256):
       6B86B273FF34FCE19D6B804EFF5A3F5747ADA4EAA22F1D49C01E52DDB7875B4B
- Id único del cheque: e542ba4d-a525-493c-8ec1-e0e31af33df1
```

Resultado de la ejecución:

```
Se ha depositado el cheque en la transacción con Id:
3FFAA94AD369CF39FD67F36C0882ACFFC003E86812FC67D57ACDFEFC31854733
Datos del cheque (Estado com.template.states.CheckDeposit):
       - Banco emisor: O=Bank1, L=Philadelpia, C=US
       - Banco depositario: O=Bank2, L=Philadelpia, C=US
       - Importe: 100
       - Moneda: EUR
       - Fecha de inicio: 2019-06-11T11:11:41.213Z
       - Fecha de caducidad: 2019-12-12T23:59:59.990Z
       - Tipo de cheque: BEARER
       - Possedor: 6B86B273FF34FCE19D6B804EFF5A3F5747ADA4EAA22F1D49C01E52DDB7875B4B
       - Id Único: e542ba4d-a525-493c-8ec1-e0e31af33df1
```

Evidencias ledger: En base al resultado de la ejecución, la ledger de Bank1 debe almacenar la nueva transacción (*3FFAA94AD369CF39FD67F36C0882ACFF…*) y marcar como consumido el estado (*com.template.states.Check*) generado en la transferencia del test 2. La ledger del Bank2 almacenará la transferencia (*3FFAA94AD369CF39FD67F36C0882ACFF…)* al ser parte de la misma. Así mismo, la ledger del Notary almacenará la transacción *E7A7E96F611AA075EE60FCB6924301469…* cuyo estado fue consumido por la transferencia.

Ledger Bank1:

```
SELECT VLS.UUID, VS.TRANSACTION_ID, VS.CONSUMED_TIMESTAMP, CONTRACT_STATE_CLASS_NAME
FROM VAULT_STATES AS VS, VAULT_LINEAR_STATES AS VLS
WHERE  VLS.UUID = 'e542ba4d-a525-493c-8ec1-e0e31af33df1' AND VLS.TRANSACTION_ID = VS.TRANSACTION_ID
ORDER BY VS.RECORDED_TIMESTAMP;
```

UUID	TRANSACTION_ID	CONSUMED_TIMESTAMP	CONTRACT_STATE_CLASS_NAME
e542ba4d-a525-493c-8ec1-e0e31af33df1	918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE	2019-06-11 13:30:21.618	com.template.states.Check
e542ba4d-a525-493c-8ec1-e0e31af33df1	E7A7E96F611AA075EE60FCB6924301469C21547496B59ECC186B1289820A8CE0	2019-06-11 13:36:15.724	com.template.states.Check
e542ba4d-a525-493c-8ec1-e0e31af33df1	3FFAA94AD369CF39FD67F36C0882ACFFC003E86812FC67D57ACDFEFC31854733	*null*	com.template.states.CheckDeposit

(3 filas, 2 ms)

Ledger Bank2:

```
SELECT VLS.UUID, VS.TRANSACTION_ID, VS.CONSUMED_TIMESTAMP, CONTRACT_STATE_CLASS_NAME
FROM VAULT_STATES AS VS, VAULT_LINEAR_STATES AS VLS
WHERE  VLS.UUID = 'e542ba4d-a525-493c-8ec1-e0e31af33df1' AND VLS.TRANSACTION_ID = VS.TRANSACTION_ID
ORDER BY VS.RECORDED_TIMESTAMP;
```

UUID	TRANSACTION_ID	CONSUMED_TIMESTAMP	CONTRACT_STATE_CLASS_NAME
e542ba4d-a525-493c-8ec1-e0e31af33df1	3FFAA94AD369CF39FD67F36C0882ACFFC003E86812FC67D57ACDFEFC31854733	*null*	com.template.states.CheckDeposit

(1 fila, 1 ms)

Ledger Notary:

```
select * from NODE_TRANSACTIONS where (tx_id = '918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE' or
tx_id = 'E7A7E96F611AA075EE60FCB6924301469C21547496B59ECC186B1289820A8CE0' or
tx_id = '3FFAA94AD369CF39FD67F36C0882ACFFC003E86812FC67D57ACDFEFC31854733');
```

TX_ID	TRANSACTION_VALUE
918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE	636f7264610100000080c562000000000001d000002a6d00000003
E7A7E96F611AA075EE60FCB6924301469C21547496B59ECC186B1289820A8CE0	636f7264610100000080c562000000000001d000002ec00000003

(2 filas, 10 ms)

- **Test 4:**

Objetivo: Hacer efectivo el depósito del cheque con Id *e542ba4d-a525-493c-8ec1-e0e31af33df1* iniciado desde Bank2.

Datos de entrada:

```
- Nodo Corda desde el que se depositará el cheque (Bank1 (1) or Bank2 (2)): 2
- Banco depositario del cheque Bank1 (1) or Bank2 (2): 2
- Referencia de operación de depósito: 1
       Poseedor en base a la referencia (Resultado SHA-256):
       6B86B273FF34FCE19D6B804EFF5A3F5747ADA4EAA22F1D49C01E52DDB7875B4B

- Id único del cheque: e542ba4d-a525-493c-8ec1-e0e31af33df1
```

Resultado de la ejecución:

```
Se ha hecho efectivo el depósito del cheque e542ba4d-a525-493c-8ec1-e0e31af33df1 en
la transacción con Id:
24C600A1367B5AFC4D600962F357B5D26C7898DC23C65073064FDD6E33AA296D
```

Evidencias ledger: En base al resultado de la ejecución, la ledger de Bank1 y Bank2 deben marcar como consumido el estado (*com.template.states.CheckDeposit*) que consumió esta transacción y que fue generado en el depósito del test 3. Así mismo, la ledger del Notary almacenará la transacción *3FFAA94AD369CF39FD67F36C0882ACFF…* cuyo estado fue consumido por la transferencia.

Ledger Bank1:

```
SELECT VLS.UUID, VS.TRANSACTION_ID, VS.CONSUMED_TIMESTAMP, CONTRACT_STATE_CLASS_NAME
FROM VAULT_STATES AS VS, VAULT_LINEAR_STATES AS VLS
WHERE  VLS.UUID = 'e542ba4d-a525-493c-8ec1-e0e31af33df1' AND VLS.TRANSACTION_ID = VS.TRANSACTION_ID
ORDER BY VS.RECORDED_TIMESTAMP;
```

UUID	TRANSACTION_ID	CONSUMED_TIMESTAMP	CONTRACT_STATE_CLASS_NAME
e542ba4d-a525-493c-8ec1-e0e31af33df1	918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE	2019-06-11 13:30:21.618	com.template.states.Check
e542ba4d-a525-493c-8ec1-e0e31af33df1	E7A7E96F611AA075EE60FCB6924301469C21547496B59ECC186B1289820A8CE0	2019-06-11 13:36:15.724	com.template.states.Check
e542ba4d-a525-493c-8ec1-e0e31af33df1	3FFAA94AD369CF39FD67F36C0882ACFFC003E86812FC67D57ACDFEFC31854733	2019-06-11 13:41:54.167	com.template.states.CheckDeposit

(3 filas, 1 ms)

Ledger Bank2:

```
SELECT VLS.UUID, VS.TRANSACTION_ID, VS.CONSUMED_TIMESTAMP, CONTRACT_STATE_CLASS_NAME
FROM VAULT_STATES AS VS, VAULT_LINEAR_STATES AS VLS
WHERE  VLS.UUID = 'e542ba4d-a525-493c-8ec1-e0e31af33df1' AND VLS.TRANSACTION_ID = VS.TRANSACTION_ID
ORDER BY VS.RECORDED_TIMESTAMP;
```

UUID	TRANSACTION_ID	CONSUMED_TIMESTAMP	CONTRACT_STATE_CLASS_NAME
e542ba4d-a525-493c-8ec1-e0e31af33df1	3FFAA94AD369CF39FD67F36C0882ACFFC003E86812FC67D57ACDFEFC31854733	2019-06-11 13:41:53.814	com.template.states.CheckDeposit

(1 fila, 1 ms)

Ledger Notary:

```
select * from NODE_TRANSACTIONS where (tx_id = '918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE' or
tx_id = 'E7A7E96F611AA075EE60FCB6924301469C21547496B59ECC186B1289820A8CE0' or
tx_id = '3FFAA94AD369CF39FD67F36C0882ACFFC003E86812FC67D57ACDFEFC31854733' or
tx_id = '24C600A1367B5AFC4D600962F357B5D26C7898DC23C65073064FDD6E33AA296D');
```

TX_ID	TRANSACTION_VALUE
3FFAA94AD369CF39FD67F36C0882ACFFC003E86812FC67D57ACDFEFC31854733	636f7264610100000080c562000000000001d0000030ff000000
918D43E086821CB482DC9AF8DA5B939A8EE13434634527BC35A833386CDC2BDE	636f7264610100000080c562000000000001d000002a6d000000
E7A7E96F611AA075EE60FCB6924301469C21547496B59ECC186B1289820A8CE0	636f7264610100000080c562000000000001d000002ec000000

(3 filas, 8 ms)

- **Test 5:**

Objetivo: Transferir dos veces consecutivas el cheque con Id único *4bf3f752-2601-40df-8755-bfbe4758cd12.*

Datos de entrada:

```
Transferencia 1:
- Nodo Corda desde el que transferir el cheque (Bank1 (1) or Bank2 (2)): 1
- Banco emisor del cheque Bank1 (1) or Bank2 (2): 1
- Teléfono del poseedor actual del cheque: 630519103
- Clave secreta del poseedor actual del cheque: oscar
        Poseedor actual del cheque (Resultado SHA-256):
        0F42DADE374186C6236F7334D230E401BCD3BF0ECC425AA7750E3F6D933DACC9
- Teléfono del usuario al que se le quiere entregar el cheque: 630630630
- Clave secreta del usuario al que se le quiere entregar el cheque: pepe
        Futuro poseedor del cheque (Resultado SHA-256):
        77BD964E201D01C42C330BCF8A3F45C2C0B955DA518C04B73A239627C16328B7
- Id único del cheque: 4bf3f752-2601-40df-8755-bfbe4758cd12
Transferencia 2:
- Nodo Corda desde el que transferir el cheque (Bank1 (1) or Bank2 (2)): 1
- Banco emisor del cheque Bank1 (1) or Bank2 (2): 1
- Teléfono del poseedor actual del cheque: 630630630
- Clave secreta del poseedor actual del cheque: pepe
        Poseedor actual del cheque (Resultado SHA-256):
        77BD964E201D01C42C330BCF8A3F45C2C0B955DA518C04B73A239627C16328B7
- Teléfono del usuario al que se le quiere entregar el cheque: 699699699
- Secret key del usuario al que se le quiere entregar el cheque: jose
        Futuro poseedor del cheque (Resultado SHA-256):
        8ADD2536D82C66C7E7F7440B70802B7D3855A391650131DEAD69B2A75434A364
- Id único del cheque: 4bf3f752-2601-40df-8755-bfbe4758cd12
```

Resultado de la ejecución:

```
Resultado transferencia 1:
Se ha transferido el cheque en la transacción con Id:
CBC2D29A6D886957EBADE58062168F25ABCC00A41B341AA70FBE402BC6B58B16
Datos del cheque (Estado com.template.states.Check):
        - Banco emisor: O=Bank1, L=Philadelpia, C=US
        - Importe: 100
        - Moneda: EUR
        - Fecha de inicio: 2019-06-11T11:48:40.393Z
        - Fecha de caducidad: 2019-12-12T23:59:59.990Z
        - Tipo de cheque: BEARER
        - Poseedor: 77BD964E201D01C42C330BCF8A3F45C2C0B955DA518C04B73A239627C16328B7
        - Id Único: 4bf3f752-2601-40df-8755-bfbe4758cd12
Resultado transferencia 2:
Se ha transferido el cheque en la transacción con Id:
533AD48C414037595E923C760A2371AAB5DD258FA54E0BB9278F346B98595965
Datos del cheque (Estado com.template.states.Check):
        - Banco emisor: O=Bank1, L=Philadelpia, C=US
        - Importe: 100
        - Moneda: EUR
        - Fecha de inicio: 2019-06-11T11:48:40.393Z
        - Fecha de caducidad: 2019-12-12T23:59:59.990Z
        - Tipo de cheque: BEARER
        - Poseedor: 8ADD2536D82C66C7E7F7440B70802B7D3855A391650131DEAD69B2A75434A364
        - Id Único: 4bf3f752-2601-40df-8755-bfbe4758cd12
```

Evidencias ledger: En base al resultado de la ejecución, la ledger de Bank1 y Bank2 deben marcar como consumido el estado (*com.template.states.CheckDeposit*) que consumió esta transacción y que fue generado en el depósito del test 3. Así mismo, la ledger del Notary almacenará la transacción *3FFAA94AD369CF39FD67F36C0882ACFF…* cuyo estado fue consumido por la transferencia.

Ledger Bank1:

```
SELECT VLS.UUID, VS.TRANSACTION_ID, VS.CONSUMED_TIMESTAMP, CONTRACT_STATE_CLASS_NAME
FROM VAULT_STATES AS VS, VAULT_LINEAR_STATES AS VLS
WHERE  VLS.UUID = '4bf3f752-2601-40df-8755-bfbe4758cd12' AND VLS.TRANSACTION_ID = VS.TRANSACTION_ID
ORDER BY VS.RECORDED_TIMESTAMP;
```

UUID	TRANSACTION_ID	CONSUMED_TIMESTAMP	CONTRACT_STATE_CLASS_NAME
4bf3f752-2601-40df-8755-bfbe4758cd12	8284B661C97AAAE47D2ED72818EA10D87819F9E96F44630BB8AF3999281443B2	2019-06-11 13:52:21.145	com.template.states.Check
4bf3f752-2601-40df-8755-bfbe4758cd12	CBC2D29A6D886957EBADE58062168F25ABCC00A41B341AA70FBE402BC6B58B16	2019-06-11 13:55:43.58	com.template.states.Check
4bf3f752-2601-40df-8755-bfbe4758cd12	533AD48C414037595E923C760A2371AAB5DD258FA54E0BB9278F346B98595965	*null*	com.template.states.Check

(3 filas, 2 ms)

Ledger Notary:

```
select * from NODE_TRANSACTIONS where (tx_id = '8284B661C97AAAE47D2ED72818EA10D87819F9E96F44630BB8AF3999281443B2' or
tx_id = 'CBC2D29A6D886957EBADE58062168F25ABCC00A41B341AA70FBE402BC6B58B16' or
tx_id = '533AD48C414037595E923C760A2371AAB5DD258FA54E0BB9278F346B98595965');
```

TX_ID	TRANSACTION_VALUE
8284B661C97AAAE47D2ED72818EA10D87819F9E96F44630BB8AF3999281443B2	636f7264610100000080c562000000000001d000002a6d0000000300a
CBC2D29A6D886957EBADE58062168F25ABCC00A41B341AA70FBE402BC6B58B16	636f7264610100000080c562000000000001d000002ec00000000300a

(2 filas, 7 ms)

- ## Test 6:

Objetivo: Evitar la transferencia de un cheque que ya ha sido entregado anteriormente. Se prueba que el usuario con teléfono *630519103* y clave secreta *oscar* que ya entrego el cheque al usuario con teléfono *630630630* no puede volver a transferir el cheque con Id *4bf3f752-2601-40df-8755-bfbe4758cd12* a otro usuario.

Datos de entrada:

```
- Nodo Corda desde el que transferir el cheque (Bank1 (1) or Bank2 (2)): 1
- Banco emisor del cheque Bank1 (1) or Bank2 (2): 1
- Teléfono del poseedor actual del cheque: 630519103
- Clave secreta del poseedor actual del cheque: oscar
      Poseedor actual del cheque (Resultado SHA-256):
      0F42DADE374186C6236F7334D230E401BCD3BF0ECC425AA7750E3F6D933DACC9
- Teléfono del usuario al que se le quiere entregar el cheque: 111222333
- Secret key del usuario al que se le quiere entregar el cheque: juan
      Futuro poseedor del cheque (Resultado SHA-256):
      C7759481271E5229B58D9CA3F4D1FB75695CD5381DFD0720264B81FEECDF3773
- Id único del cheque: 4bf3f752-2601-40df-8755-bfbe4758cd12
```

Resultado de la ejecución:

```
Node 'O=Bank1, L=Philadelpia, C=US' says: There is not any unconsumed state Check
with the Hash 0F42DADE374186C6236F7334D230E401BCD3BF0ECC425AA7750E3F6D933DACC9 and
ChId 4bf3f752-2601-40df-8755-bfbe4758cd12 in the vault states
```

No se adjunta ninguna evidencia de las ledger de Bank1 y del notario al no haberse visto actualizadas dichas ledger como consecuencia de la transacción.

- **Test 7:**

Objetivo: Detectar el intento de un banco de emitir un cheque en nombre de otro.

Datos de entrada:

```
- Nodo Corda desde el que emitir el cheque (Bank1 (1) or Bank2 (2)): 1
- Banco emisor del cheque Bank1 (1) or Bank2 (2): 2
- Importe (Número entero): 200
- Moneda (ISO 4217): eur
- Fecha caducidad (Formato: yyyy-MM-dd): 2019-12-21
- Teléfono del cliente: 630519103
- Clave secreta del cliente: oscar
       Poseedor del cheque (Resultado SHA-256):
       0F42DADE374186C6236F7334D230E401BCD3BF0ECC425AA7750E3F6D933DACC9
```

Resultado de la ejecución:

```
Node 'O=Bank1, L=Philadelpia, C=US' says: I can not issue a bearer check on behalf of
O=Bank2, L=Philadelpia, C=US
```

No se adjunta ninguna evidencia de la ledger de Bank1 al no haberse visto actualizada como consecuencia de la transacción.

- **Test 8:**

Objetivo: Atomicidad en la transferencia de un cheque. El test comprueba que si durante la transferencia el banco emisor o el notario dejasen de estar disponibles la transacción no quedaría a medias o se llegaría a completar en la otra parte sí disponible.

Datos de entrada:

```
- Nodo Corda desde el que transferir el cheque (Bank1 (1) or Bank2 (2)): 1
- Banco emisor del cheque Bank1 (1) or Bank2 (2): 1
- Teléfono del poseedor actual del cheque: 630519103
- Clave secreta del poseedor actual del cheque: oscar
       Poseedor actual del cheque (Resultado SHA-256):
       0F42DADE374186C6236F7334D230E401BCD3BF0ECC425AA7750E3F6D933DACC9
- Teléfono del usuario al que se le quiere entregar el cheque: 630630630
- Clave secreta del usuario al que se le quiere entregar el cheque: pepe
       Futuro poseedor del cheque (Resultado SHA-256):
       77BD964E201D01C42C330BCF8A3F45C2C0B955DA518C04B73A239627C16328B7
- Id único del cheque: 56010225-19c9-4cdf-8c9c-6d1623dac5ab
```

Resultado de la ejecución: La aplicación arroja una excepción no controlada pasado un tiempo. Esto se debe a que al no haberse definido timeouts en la CorDapp la aplicación no devuelve resultado alguno. Queda pendiente, por tanto, el controlar las sesiones entre nodos para evitar que la indisponibilidad de un nodo haga esperar al otro más tiempo del necesario.

- **Test 9:**

Objetivo: Demostrar que los estados (*com.template.states.Check* y *com.template.states.CheckDeposit*) almacenados en la ledger no muestran ningún dato que permita identificar a los usuarios que participan en una transacción.

Para esta demostración ejecutaremos el comando run *vaultQuery contractStateType: com.template.states.Check* directamente sobre la JVM donde se arrancó el nodo Corda de, por ejemplo, Bank1. Como resultado el nodo mostrará por consola todos los estados almacenados en su ledger que son del tipo *com.template.states.Check.* Se adjunta a continuación uno de los estados recuperados.

Resultado de la ejecución:

```
- state:
    data: !<com.template.states.Check>
      issuerBank: "O=Bank1, L=Philadelpia, C=US"
      amount: 100
      currencyCode: "EUR"
      initDate: "2019-06-11T11:48:40.393Z"
      expDate: "2019-12-12T23:59:59.990Z"
      type: "BEARER"
      owner: "8ADD2536D82C66C7E7F7440B70802B7D3855A391650131DEAD69B2A75434A364"
      linearId:
        externalId: null
        id: "4bf3f752-2601-40df-8755-bfbe4758cd12"
    contract: "com.template.contracts.ContractCheck"
    notary: "O=Notary, L=New York, C=US"
    encumbrance: null
    constraint: !<net.corda.core.contracts.SignatureAttachmentConstraint>
      key: "aSq9DsNNvGhYxYyqA9wd2eduEAZ5AXWgJTbTEw3G5d2maAq8vtLE4kZHgCs5jcB1N31cx1h
tcbufGUchxredBb6"
  ref:
    txhash: "533AD48C414037595E923C760A2371AAB5DD258FA54E0BB9278F346B98595965"
    index: 0
```

Visto los resultados que devuelve el comando se comprueba que el nodo tendría acceso a los datos del estado con lo que podría leer y recuperar cualquier de la clase que almacena. No obstante, al estar el campo *owner* (poseedor) encriptado no será posible que un banco llegue a identificar a los

usuarios. Se necesitaría un ataque de fuerza bruta sobre el campo poseedor para que el banco obtuviera el teléfono del usuario y pudiera identificarlo.

5. Planificación

Como en todo proyecto antes de su inicio se hace un estudio preliminar del mismo en base a los requerimientos establecidos por el cliente, en este caso el tutor del Trabajo Final de Máster. En base a dicho estudio se pudo hacer una planificación en la que se definieron las diferentes tareas a ejecutar y el coste temporal que cada una de ellas supondría.

Esta planificación fue inicial, y como tal, se hizo en base a una estimación de horas que se podían dedicar al proyecto y teniendo en cuenta la fecha límite de entrega fijada (4 de junio). Ante este escenario cualquier circunstancia que supusiese un recorte de horas de dedicación al proyecto o un retraso en la finalización de alguna de las tareas conllevaría una reasignación de horas en las tareas pendientes.

Aunque durante el estudio preliminar del proyecto ya se detectaron algunos puntos críticos que podrían suponer una demora de la tarea en la que se encuadraban, se identificó el desarrollo de la CorDapp como el punto más crítico ya que podría hacer retrasar e incluso no poder presentar el proyecto. Esto se debía a la nula experiencia en el desarrollo de este tipo de aplicaciones y al uso de tecnologías con las que no se está familiarizado.

También, en el estudio preliminar, se informó que solo se entregaría la aplicación desarrollada, proyecto *TFM-PoC_CordaNode* de la PoC, cuando esta cumpliese unos objetivos mínimos y siempre bajo la aceptación del cliente (tutor). En caso de no ser posible la entrega, en la memoria del proyecto se recogerían, de forma teórica, las principales características del esquema, así como los mecanismos que debería implementar para solventar algunos de los problemas que pudiesen surgir.

En un proyecto encargado por un cliente a una empresa esta debería adjuntar a la planificación un presupuesto económico en el que se tuvieran en cuenta:

- Personal necesario para el desarrollo.
- Coste del personal según categoría.
- Horas de asignación al proyecto.
- Recursos adicionales para el desarrollo.

En este proyecto, al tratarse de un Trabajo Final de Máster, no se adjunta presupuesto alguno ya que ha sido el alumno el único personal necesario y no se han necesitado recursos adicionales.

Se ha usado la herramienta Microsoft Project Professional 2016 para la elaboración de los diagramas de Gantt que se adjuntan en cada uno de los apartados.

5.1 Planificación inicial

En este punto se declaran todas las tareas que inicialmente se detectaron que se iban a llevar a cabo durante la realización del Trabajo Final de Máster.

En la planificación se fijaron unos hitos que se corresponden con las fechas máximas fijadas por la dirección del Máster para la entrega de la memoria y la presentación.

Aunque en la planificación se intentó solapar el menor número de tareas posibles, la naturaleza y el objetivo de algunas de ellas hace posible que sí se puedan simultanear. Un ejemplo de estas serían las tareas que prueban la aplicación y revisan el código ya que ambas tienen como objetivo común el mejorar la aplicación con la detección y corrección de bugs.

Para la planificación se hizo una estimación sobre el tiempo disponible a emplear en el proyecto, de tal forma que al final se computaron un total de 20h/semana:

- Lunes a viernes 2h/día
- Sábado y domingo 5h/día

Debido a la necesidad de simultanear el desarrollo del proyecto con la vida personal y laboral no se hizo una estimación superior al considerar que en algunas ocasiones no se podrían cumplir esas 20h/semana.

Al tratarse de una planificación inicial, la estimación temporal de las tareas se hizo sin tener en cuenta los posibles imprevistos que pudiesen surgir. Las horas de retraso que se puedan acumular debido a cualquier imprevisto siempre tendrían que ser compensadas con horas de días festivos y de fines de semana.

El desarrollo del proyecto conlleva la vinculación de algunas tareas con otras, esto implica la necesidad de terminar unas para empezar otras y establece una cierta dependencia que hace que el retraso de una tarea se vea reflejado en un conjunto de tareas posteriores.

La siguiente figura muestra todas las tareas que se detectaron como necesarias para la ejecución del proyecto, la asignación temporal que se le dio a cada una y las posibles vinculaciones existentes:

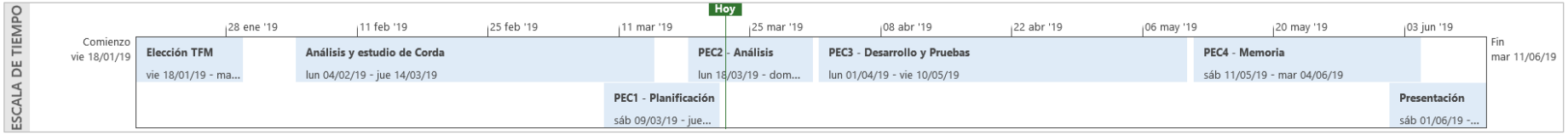


Figura 25. Escala de tiempo. Planificación inicial

DIAGRAMA DE GANTT

	Nombre de tarea	Comienzo	Fin	Predecesoras	Nombres de los recursos
1	Elección TFM	vie 18/01/19	lun 28/01/19		
2	Análisis y estudio de Corda	lun 04/02/19	mié 13/03/19	1	
3	**PEC1 - Planificación**	**sáb 09/03/19**	**jue 21/03/19**	**2**	
4	Análisis de objetivos	sáb 09/03/19	sáb 16/03/19		
5	Elaboración Plan de Trabajo	dom 17/03/19	mié 20/03/19		
6	**PEC2 - Análisis**	**lun 18/03/19**	**dom 31/03/19**	**3**	
7	Análisis blockchains existentes	lun 18/03/19	mié 27/03/19		
8	Redacción análisis y comparación blockchains (Memoria)	vie 29/03/19	dom 31/03/19	7	
9	Requisitos implantación nodo Corda	sáb 23/03/19	dom 24/03/19		
10	Estudio CorDapp	lun 25/03/19	mar 26/03/19		
11	Análisis aplicación CorDapp	jue 28/03/19	dom 31/03/19		
12	Preparación de sistemas y entornos	sáb 30/03/19	dom 31/03/19		
13	**PEC3 - Desarrollo y Pruebas**	**lun 01/04/19**	**vie 10/05/19**	**6**	
14	Instalación nodos	lun 01/04/19	dom 07/04/19		
15	Creación red Corda	lun 08/04/19	dom 14/04/19	14	
16	Desarrollo y pruebas CorDapp	lun 15/04/19	dom 28/04/19		
17	Instalación CorDapp en el nodo	lun 29/04/19	mar 30/04/19	14;16	
18	Pruebas CorDapp instalada	jue 02/05/19	lun 06/05/19	17	
19	Revisión de código	jue 02/05/19	mar 07/05/19	18CC	
20	Creación pool de pruebas	jue 02/05/19	mar 07/05/19	18CC	
21	Preparación entregables	jue 09/05/19	jue 09/05/19	19	
22	**PEC4 - Memoria**	**sáb 11/05/19**	**mar 04/06/19**	**13**	
23	Redacción Memoria TFM	sáb 11/05/19	dom 26/05/19		
24	Correción y revisión final	lun 27/05/19	lun 03/06/19	23	
25	**Presentación**	**sáb 01/06/19**	**mar 11/06/19**		
26	Redacción	sáb 01/06/19	sáb 08/06/19		
27	Corrección y revisión	dom 09/06/19	lun 10/06/19	26	
28	Entrega	mar 11/06/19	mar 11/06/19	27	
29	**Comunicación Tutor TFM**	**mar 19/03/19**	**mar 11/06/19**		
30	Comunicación Tutor TFM 1	mar 19/03/19	mar 19/03/19		

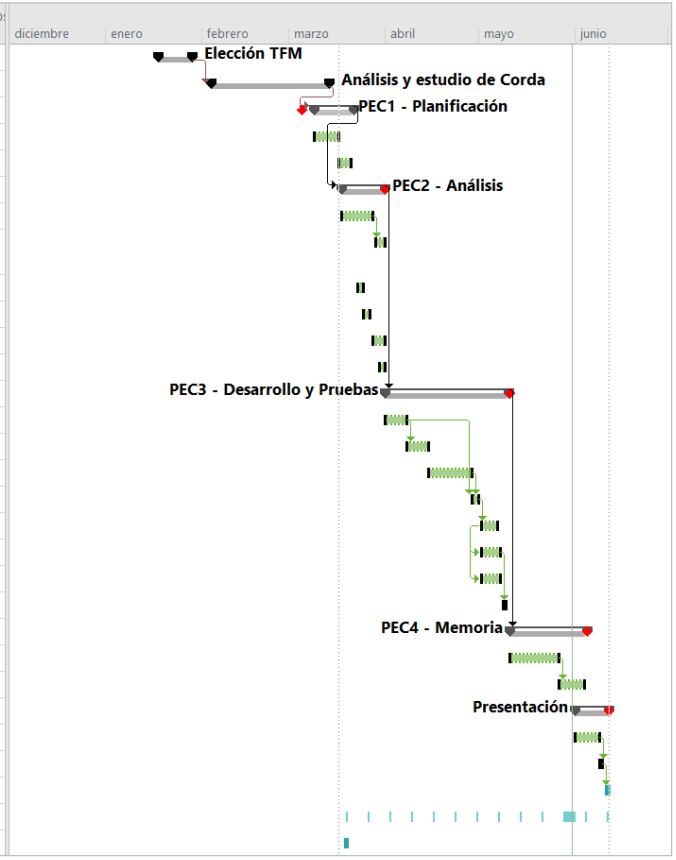


Figura 26. Diagrama de Gantt. Planificación inicial

5.2 Planificación final

Durante el desarrollo del proyecto muchas de las tareas que se habían definido en la planificación inicial no se llevaron a cabo y en otras la dedicación fue superior a la asignación temporal que se había hecho. Estas variaciones han quedado reflejadas en el diagrama de Gantt que más abajo se adjunta.

Sin duda, toda circunstancia que haga reducir la dedicación a una tarea del proyecto repercute en la calidad del Trabajo Final de Máster y, en consecuencia, en la calidad de los productos entregables.

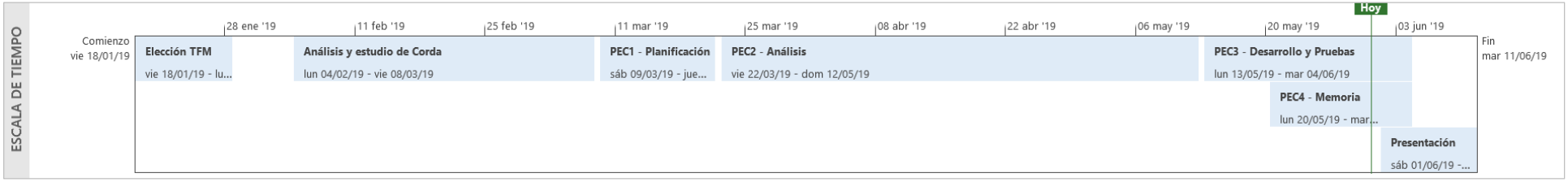


Figura 27. Escala de tiempo. Planificación final

DIAGRAMA DE GANTT

	Nombre de tarea	Comienzo	Fin	Predecesoras
1	Elección TFM	vie 18/01/19	lun 28/01/19	
2	Análisis y estudio de Corda	lun 04/02/19	vie 08/03/19	1
3	**PEC1 - Planificación**	**sáb 09/03/19**	**jue 21/03/19**	**2**
4	Análisis de objetivos	sáb 09/03/19	sáb 16/03/19	
5	Elaboración Plan de Trabajo	dom 17/03/19	jue 21/03/19	
6	**PEC2 - Análisis**	**vie 22/03/19**	**dom 12/05/19**	**3**
7	Estudio CorDapp	vie 22/03/19	sáb 13/04/19	
8	Análisis aplicación CorDapp	dom 14/04/19	vie 10/05/19	
9	Preparación de sistemas y entornos	sáb 11/05/19	dom 12/05/19	
10	**PEC3 - Desarrollo y Pruebas**	**lun 13/05/19**	**mar 04/06/19**	**6**
11	Desarrollo CorDapp	lun 13/05/19	mar 04/06/19	
12	Pruebas CordDapp	lun 27/05/19	mar 04/06/19	
13	Revisión de código	lun 27/05/19	mar 04/06/19	
14	Creación pool de pruebas	dom 02/06/19	mar 04/06/19	
15	Preparación entregables	mar 04/06/19	mar 04/06/19	
16	**PEC4 - Memoria**	**lun 20/05/19**	**mar 04/06/19**	
17	Redacción Memoria TFM	lun 20/05/19	dom 02/06/19	
18	Correción y revisión final	lun 03/06/19	mar 04/06/19	17
19	**Presentación**	**sáb 01/06/19**	**mar 11/06/19**	
20	Redacción	sáb 01/06/19	sáb 08/06/19	
21	Corrección y revisión	dom 09/06/19	lun 10/06/19	20
22	Entrega	mar 11/06/19	mar 11/06/19	21
23	**Comunicación Tutor TFM**	**mar 19/03/19**	**mar 11/06/19**	
24	Comunicación Tutor TFM 1	mar 19/03/19	mar 19/03/19	

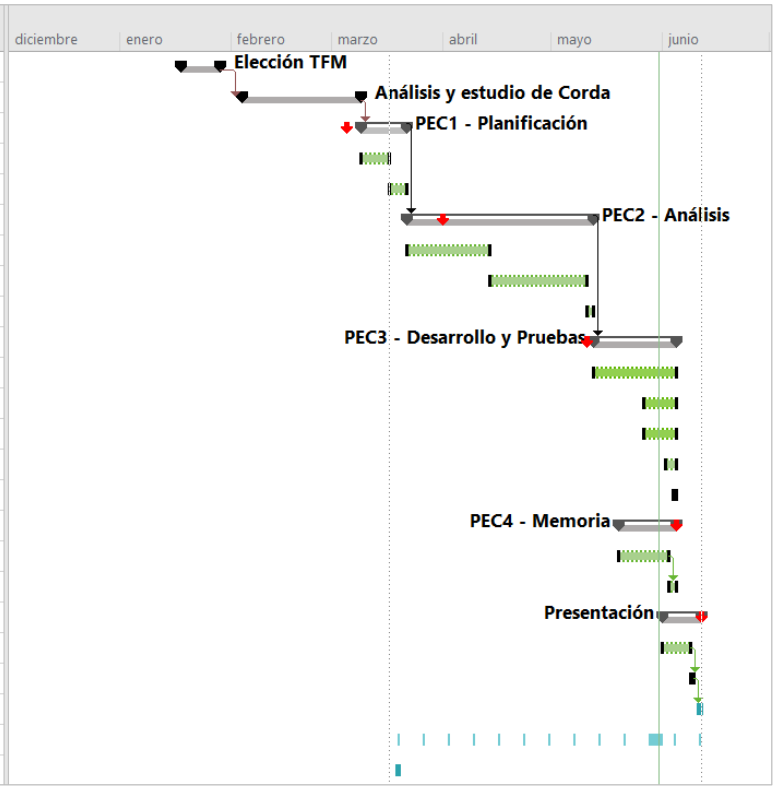


Figura 28. Diagrama de Gantt. Planificación final

6. Conclusiones

Como se ha ido detallando a lo largo de la memoria, el objetivo de este Trabajo Final de Máster ha sido el análisis, diseño y prueba de concepto de un sistema que permitiese la emisión, transferencia y depósito de cheques al portador basado en la plataforma Blockchain Corda.

Se ha visto que Corda es una plataforma en evolución que en cada nueva versión publicada añade nuevas funcionalidades y mejora todos aquellos puntos débiles o bugs que pudiesen tener las versiones anteriores. Durante el estudio de la API de Corda se ha comprobado que esta se ajusta a aplicaciones como la presentada ya que, además de ofrecer unas características funcionales que permiten una transferencia segura y anónima de un cheque al portador, detectando y evitando los intentos de hacer un doble uso de un cheque, permite la creación de una red privada en la que el acceso a la misma estará controlado y en la que cada miembro conoce al resto. Son, sin duda, estos hechos los que van a permitir a bancos una adhesión segura y confiable al sistema.

Fue en el punto 3.2 de este documento donde se definieron los requerimientos de seguridad que debía de cumplir el esquema en análisis y ha sido gracias a la PoC donde se ha podido demostrar y ratificar, de forma práctica y en base a los resultados de las pruebas realizadas, que Corda nos sirve para satisfacer todos aquellos objetivos que buscaba este proyecto.

En el esquema diseñado se han detectado ciertos puntos débiles que pueden comprometer la disponibilidad y la seguridad del sistema:

- Notario de la red Corda: Si un atacante se hiciera con el control de la máquina podría modificar su comportamiento para que la aplicación no hiciera una correcta comprobación de estados y permitiese el doble consumo. También podría provocar una denegación de servicio (*DoS*) en toda la red si deshabilita su nodo.
- Tercera Parte Confiable (*TPC*): Ya que todas las transferencias parten de ella, si un atacante se hiciera con su control podría suplantar al destinatario de las transferencias de tal modo que siempre dicho atacante fuera el receptor del cheque. Además, podría provocar una *DoS* del sistema en los procesos de transferencia.
- Comunicaciones entre bancos y entre *TPC* y bancos: Tanto el canal como los datos entre bancos y *TPC* están expuestos a ataques al viajar estos por redes públicas. Un atacante que llevase a cabo una captura de tráfico (*sniffer*) en alguna de las redes o efectuase un ataque Man In The Middle (*MitM*) sería capaz de obtener datos de los cheques y utilizarlos en beneficio propio.

La mejor forma de reducir la criticidad del notario sería creando un clúster de notarios que obligase a un ataque masivo para hacerse con el control del sistema.

Implementar sistemas de redundancia y políticas restrictivas de red y de acceso incrementará el nivel de seguridad notario.

En cuanto al *TPC*, al ser un sistema expuesto al exterior el riesgo de ser atacado es mayor. Por tanto, se hace, más importante si cabe, el añadir redundancia de sistemas y políticas de restrictivas de red y de acceso que obliguen solo a dejar accesible lo realmente necesario. También, se aplicarán medidas preventivas y reactivas ante ataques de DoS.

Para solucionar problemas de seguridad del canal de comunicaciones bastaría con establecer VPNs (Virtual Path Networks) entre bancos y *TPC*.

Desde un punto de vista personal este Trabajo Final de Máster me ha permitido investigar y descubrir una nueva plataforma Blockchain que en un futuro puede llegar a convertirse en un referente dentro del sector financiero. Aunque Corda merece mucho más tiempo de estudio, análisis e investigación para poder conocer en profundidad todo su "potencial", este proyecto sí que me ha permitido comprobar la flexibilidad que ofrece para adaptarse a un escenario que requiera seguridad y consenso en sus transacciones.

Sin duda, considero que el desarrollo de este Trabajo Final de Máster ha sido gratificante y de especial interés ya que he podido conocer una plataforma que a día de hoy cubriría algunas de las necesidades a las que me enfrento en mi actividad laboral y que son solventadas con tecnologías obsoletas y no tan seguras. Considero este proyecto como un punto de partida que sin duda me va a ayudar a crecer profesionalmente.

7. Futuras mejoras

La primera mejora empezaría por añadir nuevas funcionalidades al esquema que incrementasen su valor, algunas de estas nuevas funcionalidades podrían ser:

- División de un cheque en otros de menor importe.
- Unificación de cheques.
- Posibilidad de adjuntar documentos a la transferencia de un cheque.
- Reseteo de la clave secreta que definió el usuario para el cheque.

Las implementaciones de estas funcionalidades se harían sobre el proyecto *TFM-PoC_CordaNode* pero antes sería conveniente adaptar este a las sugerencias que hace Corda sobre el lenguaje de programación a usar y codificarlo en *Kotlin*. Una vez hecha la codificación se añadirían nuevos flujos que incluyesen las funcionalidades arriba indicadas.

En cuanto al proyecto *TFM-PoC_RPCClients,* a este se añadirían los clientes RPC necesarios para invocar de forma remota a los nuevos flujos de la CorDapp. Además, se implementaría la prueba de cocimiento nula para que pudiese ser simulada.

Fuera del ámbito de la *PoC* sería conveniente el generar un manual que incluyera los pasos a seguir para la instalación de un nodo Corda, y las correspondientes CorDapp, en un servidor dedicado.

Una última mejora a llevar a cabo sería el añadir al esquema un caso real de uso que estuviese basado en las tecnologías actuales. Este caso de uso podría hacer uso, por ejemplo, de los smartphones para, a través de una app, solicitar la emisión, transferencia y depósito de un cheque.

- En el proceso de emisión, la app redirigiría al usuario a su banco y este le generaría un cheque que quedaría almacenado de forma automática y segura en la app del usuario.
- En el proceso de transferencia, el usuario tenedor del cheque solo tendría que informar el teléfono del receptor. A partir de aquí el envío del cheque al receptor y la prueba de conocimiento nulo serían transparentes para los usuarios.
- En el proceso de depósito, la app redirigiría al usuario a su banco y este haría el ingreso del cheque en su cuenta de forma automática.

La app internamente tiene que ser capaz de generar y almacenar las claves secretas que asigna a los cheques.

En cuanto a la Tercera Parte Confiable, esta sería el servidor de la app. Tendría que ser capaz de participar en la prueba de conocimiento y establecer conexiones seguras con los bancos de adheridos a la red.

Referencias

[1] Bitcoin web, https://bitcoin.org/es/.

[2] Satoshi Nakamoto (24 de mayo de 2009). "Bitcoin: A Peer-to-Peer Electronic Cash System".

[3] Nick Szabo (1 de septiembre de 1997). "Formalizing and Securing Relationships on Public Networks". First Monday (vol.: 2, núm.: 9).

[4] Christopher D. Clack, Vikram A. Bakshi y Lee Braine (4 de agosto de 2016 – revisado 15 de marzo de 2017). "Smart Contract Templates: foundations, design landscape and research directions".

[5] Adam Back (1 de agosto de 2002). "Hashcash - A Denial of Service Counter-Measure".

[6] Stuart Haber y W. Scott Stornetta (1991). "How To Time-Stamp a Digital Document". Journal of Cryptology (vol.: 3, núm.: 2, págs.: 99-111).

[7] Deepak Puthal, Nisha Malik, Saraju P. Mohanty, Elias Kougianos y Gautam Das (2018). "Everything you Wanted to Know about the Blockchain".

[8] Karl Wüst y Arthur Gervais (2017). "Do you need a Blockchain?", Departamento de ciencias de la computación, Escuela Politécnica Federal de Zurich (Suiza).

[9] Corda web, https://www.corda.net/ y https://docs.corda.net/index.html.

[10] Richard Gendal Brown, James Carlyle, Ian Grigg, Mike Hearn (agosto 2016). "Corda: An Introduction"

[11] Richard Gendal Brown (mayo de 2018). "The Corda Platform: An Introduction".

[12] Mike Hearn (29 de noviembre de 2016). "Corda: A distributed ledger".

[13] Corda (9 de abril de 2019). "Corda Developer Documentation".

[14] C. P. Schnorr (1991). "Efficient Signature Generation by Smart Cards". Journal of Cryptology (vol.: 4, págs.: 161-174).

[15] Robert Cragie y Feng Hao (2018). "Elliptic Curve J-PAKE Cipher Suites for Transport Layer Security (TLS)". Universidad de NewCastle (Reino Unido).

[16] Raj Jain (2011). "Digital Signature". Universidad de Washington (EEUU).

[17] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk (mayo 2008). "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile". Internet Engineering Task Force (IETF).

[18] K. Zeilenga (junio 2006). "Lightweight Directory Access Protocol(LDAP): String Representation of Distinguished Names". Internet Engineering Task Force (IETF).