revocado, y Bob tiene las claves necesarias para emitir transacciones de penalización contra ellos, en caso de que Alice intente transmitir uno de ellos.

Alice podría tener un incentivo para hacer trampa porque todas las transacciones de compromiso anteriores le darían una mayor proporción del saldo del canal de lo que tiene derecho. Digamos, por ejemplo, que Alice intentó transmitir el Compromiso #1. Esa transacción de compromiso le pagaría a Alice 70 000 satoshis y a Bob 70 000 satoshis. Si Alice pudiera transmitir y gastar su producción to_local, estaría robando efectivamente 30,000 satoshis de Bob al revertir sus dos últimos pagos a Bob.

Alice decide correr un gran riesgo y transmitir el Compromiso #1 revocado, para robarle 30,000 satoshis a Bob. En la Figura 7-13 vemos el antiguo compromiso de Alice que transmite a la cadena de bloques de Bitcoin.

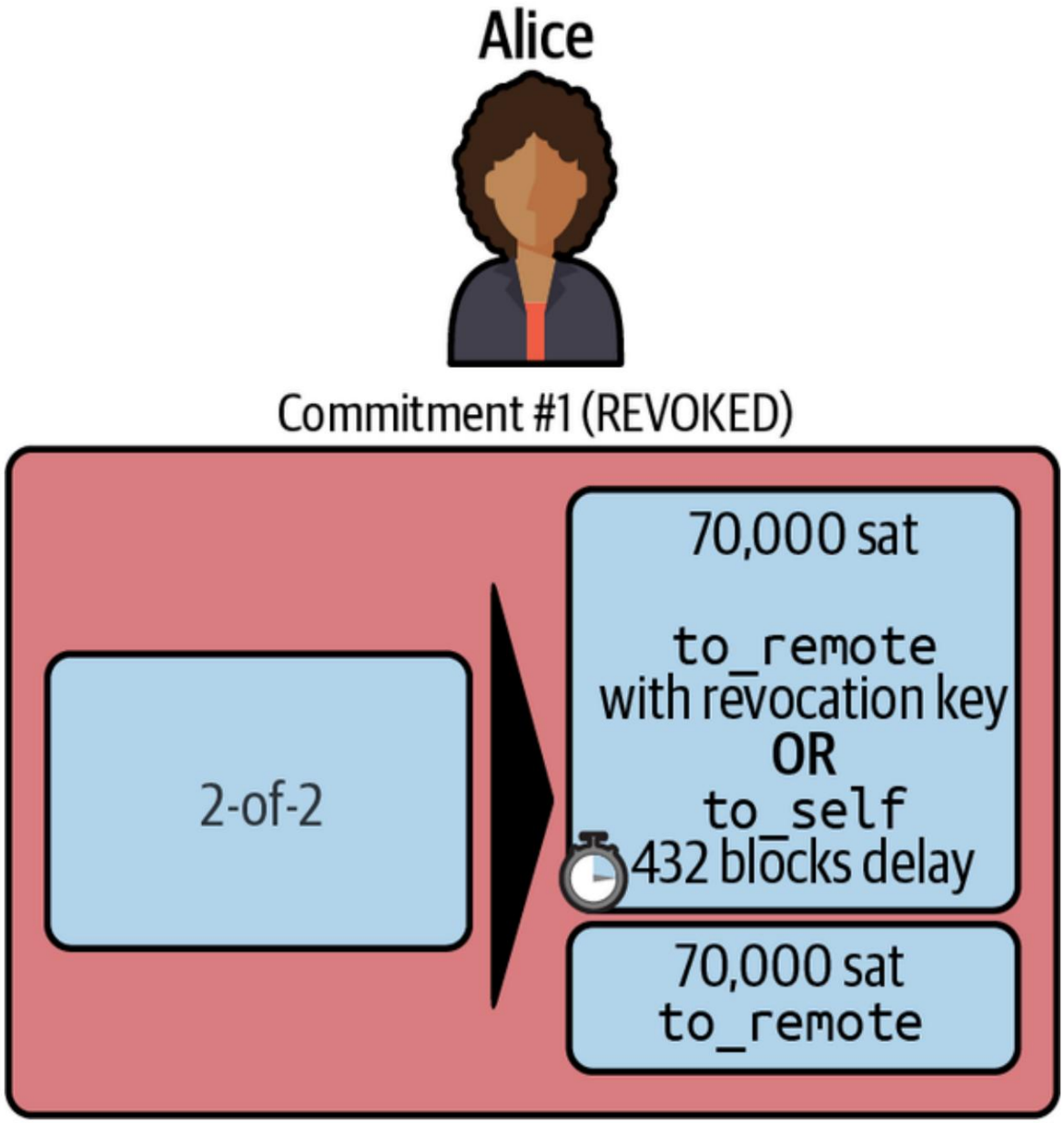

Figura 7-13. Alicia haciendo trampa

Como puede ver, el antiguo compromiso de Alice tiene dos resultados, uno pagándose ella misma 70 000 satoshis (to_local output) y otro pagándole a Bob 70 000 satoshis. Alice aún no puede gastar su producción to_local de 70ÿ000 porque tiene un bloqueo de tiempo de 432 bloques (3 días). Ahora espera que Bob no se dé cuenta durante tres días.

Desafortunadamente para Alice, el nodo de Bob está monitoreando diligentemente la cadena de bloques de Bitcoin y ve una transmisión de transacción de compromiso anterior y

(eventualmente) confirmado en la cadena.

El nodo de Bob transmitirá inmediatamente una transacción de penalización. Dado que este antiguo compromiso fue revocado por Alice, Bob tiene el per_commitment_secret que Alice le envió. Él usa ese secreto para construir una firma para revocation_pubkey. Mientras Alice tiene que esperar 432 bloques, Bob puede gastar ***ambas*** salidas inmediatamente. Puede gastar la salida to_remote con sus claves privadas porque estaba destinado a pagarle de todos modos. También puede gastar la salida destinada a Alice con una firma de la clave de revocación. Su nodo transmite la transacción de penalización que se muestra en la figura 7-14.

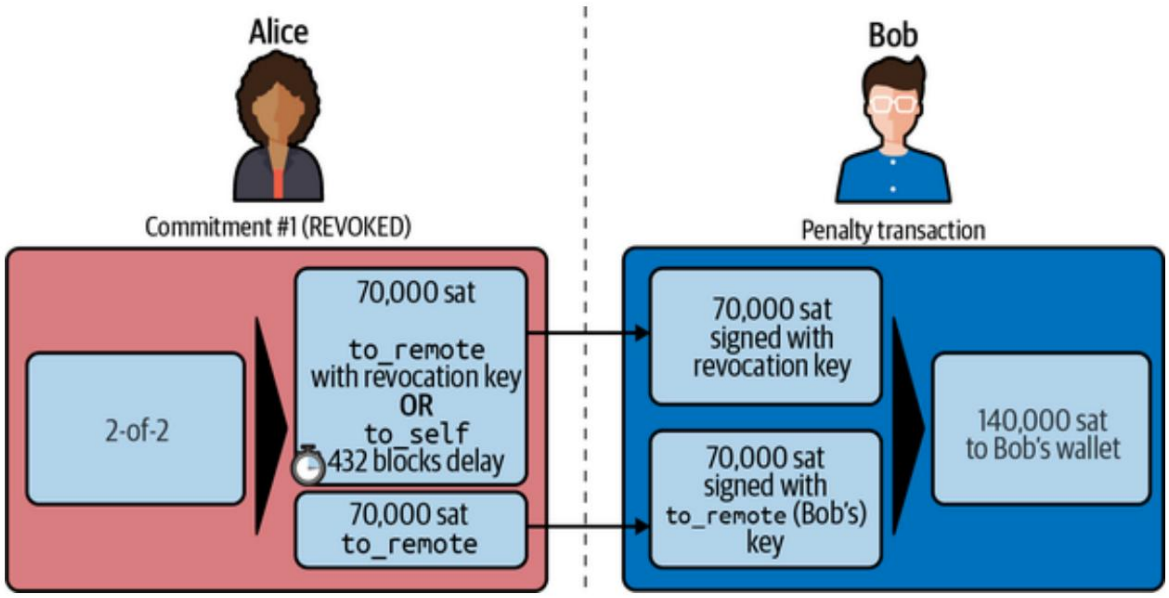

Figura 7-14. Trampa y penalización

La transacción de penalización de Bob paga 140 000 satoshis a su propia billetera, ocupando toda la capacidad del canal. Alice no solo ha fallado en hacer trampa, ¡sino que lo ha perdido todo en el intento!

The Channel Reserve: asegurando la piel en el juego

Es posible que haya notado que hay una situación especial que debe abordarse.
Si Alice pudiera seguir gastando su saldo hasta que sea cero, estaría en condiciones de cerrar el canal transmitiendo una transacción de compromiso anterior sin correr el riesgo de una penalización: ya sea el compromiso revocado

la transacción tiene éxito después de la demora, o se atrapa al tramposo, pero no hay consecuencias porque la penalización es cero. Desde la perspectiva de la teoría de juegos, es dinero gratis intentar hacer trampa en esta situación. Es por eso que la reserva del canal está en juego, por lo que un posible tramposo siempre enfrenta el riesgo de una penalización.

Cerrando el Canal (Cierre Cooperativo)

Hasta ahora hemos visto las transacciones de compromiso como una forma posible de cerrar un canal, unilateralmente. Este tipo de cierre de canal no es ideal porque fuerza un bloqueo de tiempo en el socio de canal que lo usa.

Una mejor manera de cerrar un canal es un cierre cooperativo. En un cierre cooperativo, los dos socios de canal negocian una transacción de compromiso final llamada transacción de ***cierre*** que paga a cada parte su saldo inmediatamente a la billetera de destino de su elección. Luego, el socio que inició el flujo de cierre del canal transmitirá la transacción de cierre.

El flujo de mensajes de cierre se define en el PERNO n.º 2: Protocolo de pares, cierre de canal y se muestra en la figura 7-15.

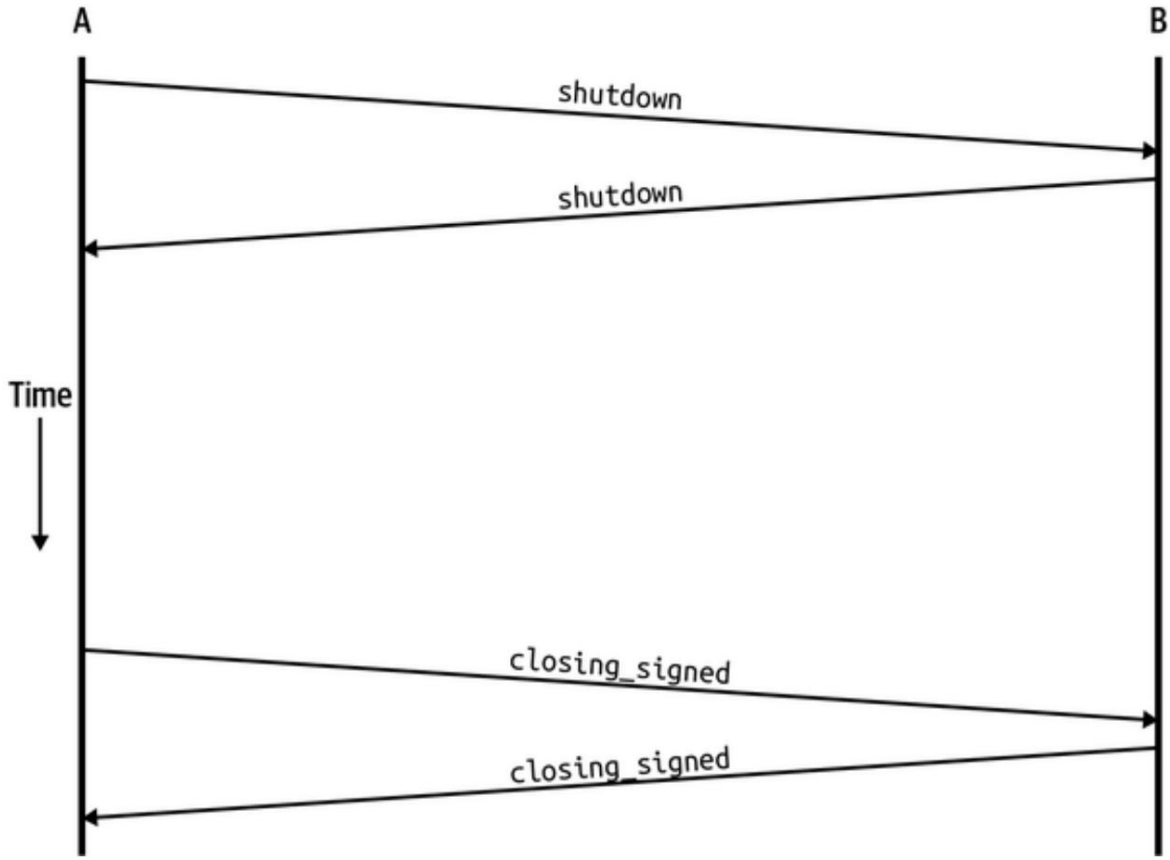

Figura 7-15. El flujo de mensajes de cierre de canal

El mensaje de apagado

El cierre del canal comienza cuando uno de los dos socios del canal envía el mensaje de apagado. El contenido de este mensaje se muestra aquí:

[id_canal:id_canal] [u16:len]
[len*byte:scriptpubkey]

Canal ID

El identificador de canal para el canal que queremos cerrar.

solamente

La longitud de la secuencia de comandos de la billetera de destino que este socio de canal quiere recibir su saldo

scriptpubkey

> Un script de Bitcoin de la billetera de destino, en uno de los "estándar" Formatos de direcciones de Bitcoin (P2PKH, P2SH, P2WPKH, P2WSH, etc.; consulte el Glosario)

Digamos que Alice envía el mensaje de apagado a Bob para cerrar su canal. Alice especificará un script de Bitcoin que corresponde a la dirección de Bitcoin de su billetera. Ella le dice a Bob: hagamos una transacción de cierre que pague mi saldo a esta billetera.

Bob responderá con su propio mensaje de apagado que indica que acepta cerrar el canal de manera cooperativa. Su mensaje de apagado incluye el script para la dirección de su billetera.

Ahora, tanto Alice como Bob tienen la dirección de billetera preferida del otro y pueden construir transacciones de cierre idénticas para liquidar el saldo del canal.

El mensaje de cierre_firmado

Suponiendo que el canal no tenga compromisos o actualizaciones pendientes y que los socios del canal hayan intercambiado los mensajes de cierre que se muestran en la sección anterior, ahora pueden finalizar este cierre cooperativo.

El ***patrocinador*** del canal (Alice en nuestro ejemplo) comienza enviando un mensaje de cierre_firmado a Bob. Este mensaje propone una tarifa de transacción para la transacción en cadena y la firma de Alice (la multifirma 2 de 2) para la transacción de cierre. El mensaje de cierre_firmado se muestra aquí:

```
[channel_id:channel_id]
[u64:fee_satoshis] [firma:firma]
```

Canal ID

El identificador del canal

fee_satoshis

La tarifa de transacción en cadena propuesta, en satoshis

firma

La firma del remitente para la transacción de cierre.

Cuando Bob recibe esto, puede responder con un mensaje de cierre_firmado propio. Si está de acuerdo con la tarifa, simplemente devuelve la misma tarifa propuesta y su propia firma. Si no está de acuerdo, debe proponer una tarifa diferente: la tarifa de satoshis.

Esta negociación puede continuar con mensajes de cierre_firmados de ida y vuelta hasta que los dos socios de canal acuerden una tarifa.

Una vez que Alice recibe un mensaje de cierre_firmado con la misma tarifa que la que propuso en su último mensaje, la negociación está completa. Alice firma y transmite la transacción de cierre y el canal se cierra.

La transacción de cierre de la cooperativa

La transacción de cierre cooperativo se parece a la última transacción de compromiso que habían acordado Alice y Bob. Sin embargo, a diferencia de la última transacción de compromiso, no tiene bloqueos de tiempo ni claves de revocación de penalización en las salidas. Dado que ambas partes cooperan para producir esta transacción y no se comprometerán más, no hay necesidad de elementos asimétricos, demorados y revocables en esta transacción.

Por lo general, las direcciones utilizadas en esta transacción de cierre cooperativo se generan recientemente para cada canal que se cierra. Sin embargo, también es posible que ambas partes ***bloqueen*** una dirección de "entrega" que se utilizará para enviar sus fondos liquidados en forma cooperativa. Dentro del espacio de nombres TLV de los mensajes open_channel y accept_channel, ambos lados tienen la libertad de especificar un "script de apagado inicial". Comúnmente, esta dirección se deriva

de claves que residen en almacenamiento en frío. Esta práctica sirve para aumentar la seguridad de los canales: si un socio de canal es pirateado de alguna manera, entonces el pirata informático no puede cerrar el canal de manera cooperativa utilizando una dirección que controle. En cambio, el socio de canal honesto e intransigente se negará a cooperar en el cierre de un canal si no se utiliza la dirección de cierre inicial especificada. Esta función crea efectivamente un "bucle cerrado" que restringe el flujo de fondos fuera de un canal determinado.

Alice transmite una transacción que se muestra en la Figura 7-16 para cerrar el canal.

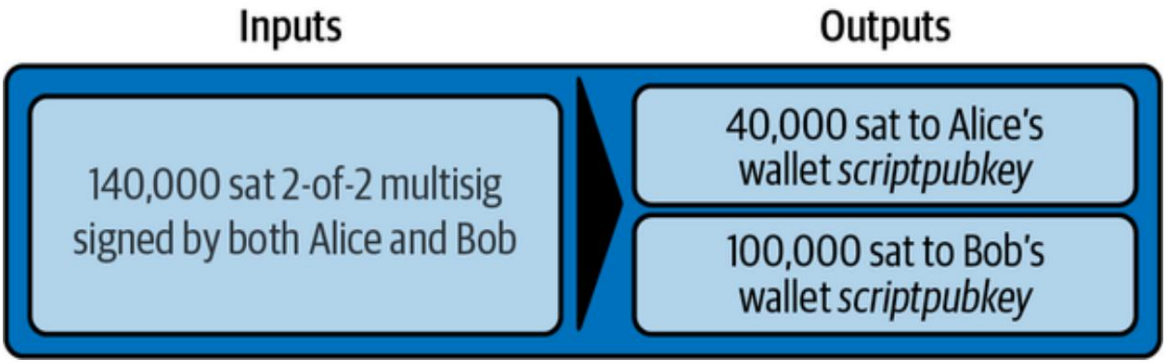

Figura 7-16. La cooperativa cierra la transacción

Tan pronto como se confirme esta transacción de cierre en la cadena de bloques de Bitcoin, el canal se cerrará. Ahora, Alice y Bob pueden gastar sus productos como les plazca.

Conclusión

En esta sección analizamos los canales de pago con mucho más detalle. Examinamos tres flujos de mensajes utilizados por Alice y Bob para negociar la financiación, los compromisos y el cierre del canal. También mostramos la estructura de las transacciones de financiamiento, compromiso y cierre, y analizamos los mecanismos de revocación y sanción.

Como veremos en los próximos capítulos, los HTLC se utilizan incluso para pagos locales entre socios de canal. No son necesarios, pero el protocolo es mucho más sencillo si los pagos locales (un canal) y enrutados (varios canales) se realizan de la misma forma.

En un canal de pago único, la cantidad de pagos por segundo solo está limitada por la capacidad de la red entre Alice y Bob. Siempre que los socios de canal puedan enviar algunos bytes de datos de un lado a otro para aceptar un nuevo saldo de canal, habrán realizado efectivamente un pago. Es por eso que podemos lograr un rendimiento de pagos mucho mayor en Lightning Network (fuera de la cadena) que el rendimiento de transacciones que puede manejar la cadena de bloques de Bitcoin (en la cadena).

En los próximos capítulos, analizaremos el enrutamiento, los HTLC y su uso en las operaciones del canal.

Capítulo 8. Enrutamiento en una red de canales de pago

En este capítulo, finalmente desglosaremos cómo se pueden conectar los canales de pago para formar una red de canales de pago a través de un proceso llamado ***enrutamiento.*** Específicamente, veremos la primera parte de la capa de enrutamiento, el protocolo de "contratos multisalto atómicos y sin confianza". Esto se destaca mediante un esquema en el conjunto de protocolos, que se muestra en la Figura 8-1.

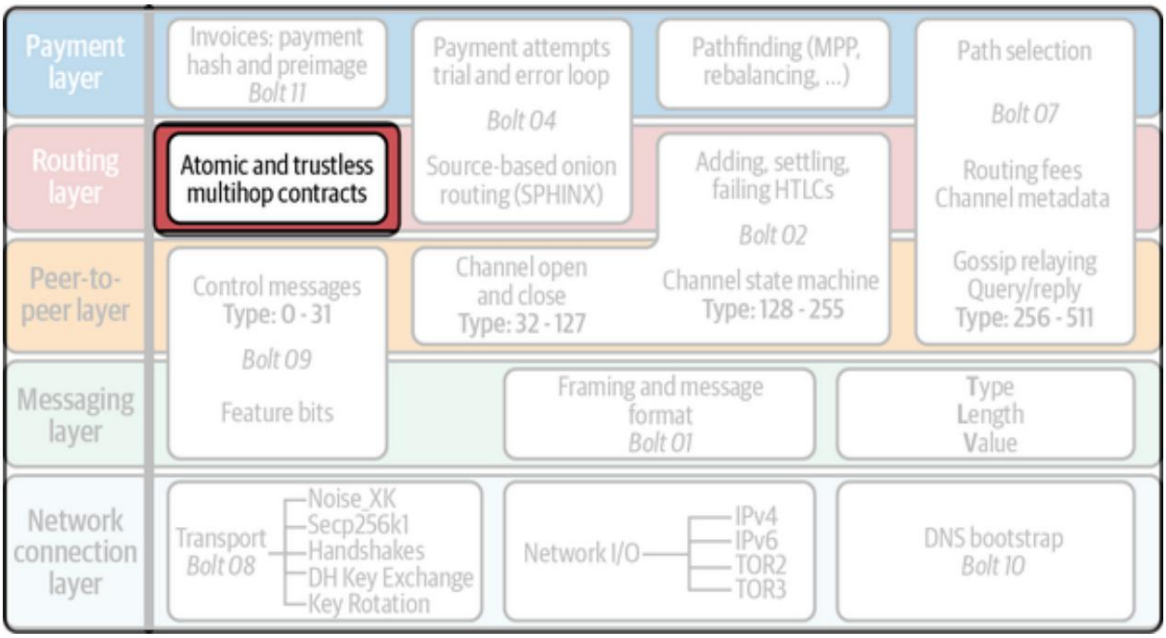

Figura 8-1. Enrutamiento de pago atómico en el conjunto de protocolos Lightning

Enrutamiento de un pago

En esta sección, examinaremos el enrutamiento desde la perspectiva de Dina, una jugadora que recibe donaciones de sus fanáticos mientras transmite sus sesiones de juego.

La innovación de los canales de pago enrutados le permite a Dina recibir propinas sin mantener un canal separado con cada uno de sus fanáticos que

quiere darle propina. Siempre que exista una ruta de canales bien financiados desde ese espectador hasta Dina, ella podrá recibir el pago de ese fan.

En la Figura 8-2 , vemos un posible diseño de red creado por varios canales de pago entre los nodos Lightning. Todos en este diagrama pueden enviarle un pago a Dina construyendo una ruta. Imagina que Fan 4 quiere enviarle un pago a Dina. ¿Ves el camino que podría permitir que eso suceda? Fan 4 podría enrutar un pago a Dina a través de Fan 3, Bob y Chan. De manera similar, Alice podría enrutar un pago a Dina a través de Bob y Chan.

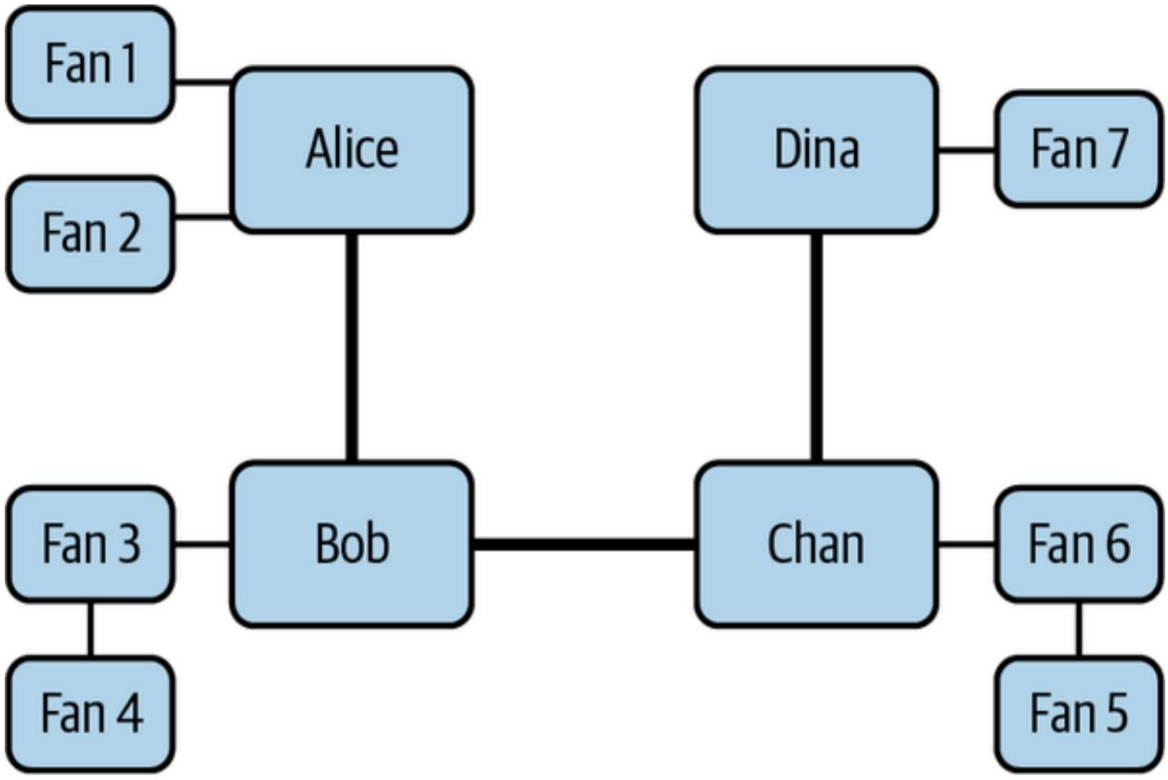

Figura 8-2. Los fanáticos se conectaron (in)directamente a Dina en Lightning Network

Los nodos a lo largo del camino desde el ventilador hasta Dina son intermediarios llamados ***nodos de enrutamiento*** en el contexto del enrutamiento de un pago. No hay diferencia funcional entre los nodos de enrutamiento y los nodos operados por los ventiladores de Dina. Cualquier nodo Lightning es capaz de enrutar pagos a través de sus canales de pago.

Es importante destacar que los nodos de enrutamiento no pueden robar los fondos mientras enrutan un pago de un fan a Dina. Además, los nodos de enrutamiento no pueden perder dinero mientras participan en el proceso de enrutamiento. Los nodos de enrutamiento pueden cobrar un

tarifa de enrutamiento por actuar como intermediario, aunque no tienen que hacerlo y pueden optar por enrutar los pagos de forma gratuita.

Otro detalle importante es que, debido al uso del enrutamiento cebolla, los nodos intermediarios solo conocen explícitamente el nodo que los precede y el nodo que los sigue en la ruta. No necesariamente sabrán quién es el remitente y el destinatario del pago. Esto permite a los fanáticos usar nodos intermediarios para pagar a Dina, sin filtrar información privada y sin correr el riesgo de robo.

Este proceso de conectar una serie de canales de pago con seguridad de extremo a extremo y la estructura de incentivos para que los nodos ***reenvíen*** pagos es una de las innovaciones clave de Lightning Network.

En este capítulo, nos sumergiremos en el mecanismo de enrutamiento en Lightning Network, detallando la forma precisa en que los pagos fluyen a través de la red. Primero, aclararemos el concepto de enrutamiento y lo compararemos con el de búsqueda de ruta, porque a menudo se confunden y se usan indistintamente.
A continuación, construiremos el protocolo de equidad: un protocolo atómico, sin confianza y de múltiples saltos que se utiliza para enrutar los pagos. Para demostrar cómo funciona este protocolo de equidad, utilizaremos un equivalente físico de transferir monedas de oro entre cuatro personas. Por último, veremos la implementación del protocolo atómico, sin confianza y multisalto que se usa actualmente en Lightning Network, que se denomina contrato hash de bloqueo de tiempo (HTLC).

Enrutamiento versus Pathfinding

Es importante tener en cuenta que separamos el concepto de ***enrutamiento*** del concepto de búsqueda de rutas . Estos dos conceptos a menudo se confunden y el término ***enrutamiento*** se usa a menudo para describir ambos conceptos. Eliminemos la ambigüedad antes de continuar.

La búsqueda de ruta , que se trata en el Capítulo 12, es el proceso de encontrar y elegir una ruta contigua hecha de canales de pago que conectan al remitente A con el destinatario B. El remitente de un pago hace la búsqueda de ruta al

examinando el ***gráfico de canal*** que han ensamblado a partir de anuncios de canal chismeados por otros nodos.

El enrutamiento se refiere a la serie de interacciones a través de la red que intentan reenviar un pago desde un punto A a otro punto B, a través de la ruta previamente seleccionada por la búsqueda de ruta. El enrutamiento es el proceso activo de enviar un pago por una ruta, lo que implica la cooperación de todos los nodos intermediarios a lo largo de esa ruta.

Una regla general importante es que es posible que exista una ***ruta*** entre Alice y Bob (quizás incluso más de una), pero puede que no haya una ***ruta*** activa por la cual enviar el pago. Un ejemplo es el escenario en el que todos los nodos que conectan a Alice y Bob están desconectados actualmente. En este ejemplo, se puede examinar el gráfico de canales y conectar una serie de canales de pago de Alice a Bob, por lo que existe una ***ruta .*** Sin embargo, debido a que los nodos intermediarios están fuera de línea, el pago no se puede enviar y, por lo tanto, no existe una ***ruta .***

Creación de una red de canales de pago

Antes de sumergirnos en el concepto de un pago atómico multisalto sin confianza, analicemos un ejemplo. Volvamos a Alice quien, en capítulos anteriores, le compró un café a Bob con quien tiene un canal abierto.
Ahora Alice está viendo una transmisión en vivo de Dina, la jugadora, y quiere enviarle a Dina una propina de 50 000 satoshis a través de Lightning Network. Pero Alice no tiene un canal directo con Dina. ¿Qué puede hacer Alicia?

Alice podría abrir un canal directo con Dina; sin embargo, eso requeriría liquidez y tarifas en cadena que podrían ser más que el valor de la propina en sí. En su lugar, Alice puede usar sus canales abiertos existentes para enviar un consejo a Dina ***sin*** necesidad de abrir un canal directamente con Dina. Esto es posible, siempre que exista alguna ruta de canales desde Alice a Dina con capacidad suficiente para enrutar la punta.

Como puede ver en la Figura 8-3, Alice tiene un canal abierto con Bob, el dueño de la cafetería. Bob, a su vez, tiene un canal abierto con el software.

el desarrollador Chan, quien lo ayuda con el sistema de punto de venta que usa en su cafetería. Chan también es propietario de una gran empresa de software que desarrolla el juego que juega Dina, y ya tienen un canal abierto que Dina utiliza para pagar la licencia del juego y los elementos del juego.

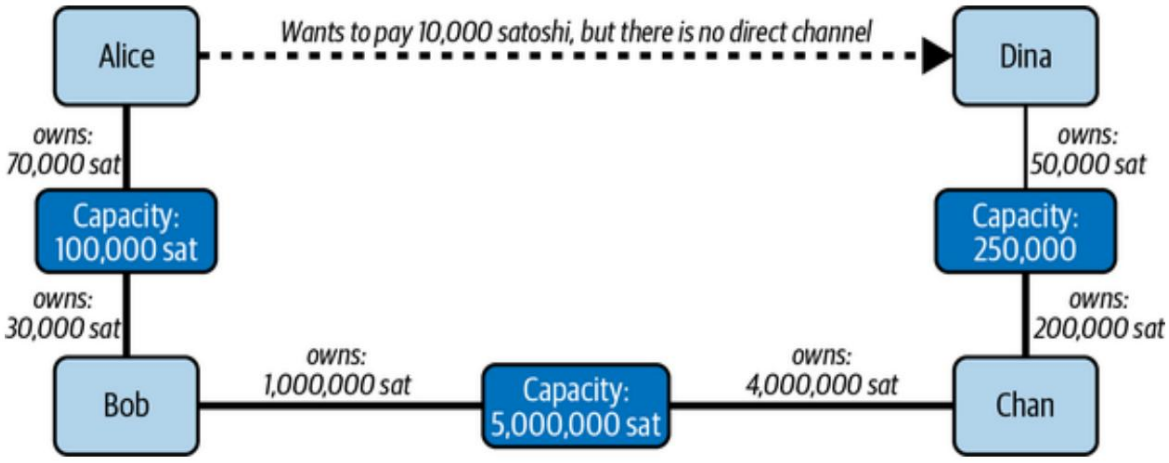

Figura 8-3. Una red de canales de pago entre Alice y Dina

Es posible rastrear una ***ruta*** desde Alice hasta Dina que utilice a Bob y Chan como nodos de enrutamiento intermediarios. Luego, Alice puede crear una ***ruta*** a partir de este camino delineado y usarla para enviar una propina de unos miles de satoshis a Dina, y Bob y Chan ***reenviarán*** el pago. Esencialmente, Alice le pagará a Bob, quien le pagará a Chan, quien le pagará a Dina. No se requiere un canal directo de Alice a Dina.

El principal desafío es hacer esto de una manera que evite que Bob y Chan roben el dinero que Alice quiere que se le entregue a Dina.

Un ejemplo físico de "enrutamiento"

Para comprender cómo Lightning Network protege el pago mientras se enruta, podemos compararlo con un ejemplo de enrutamiento de pagos físicos con monedas de oro en el mundo real.

Supongamos que Alice quiere darle 10 monedas de oro a Dina, pero no tiene acceso directo a Dina. Sin embargo, Alice conoce a Bob, quien conoce a Chan, quien conoce a Dina, por lo que decide pedir ayuda a Bob y Chan. Esto se muestra en la Figura 8-4.

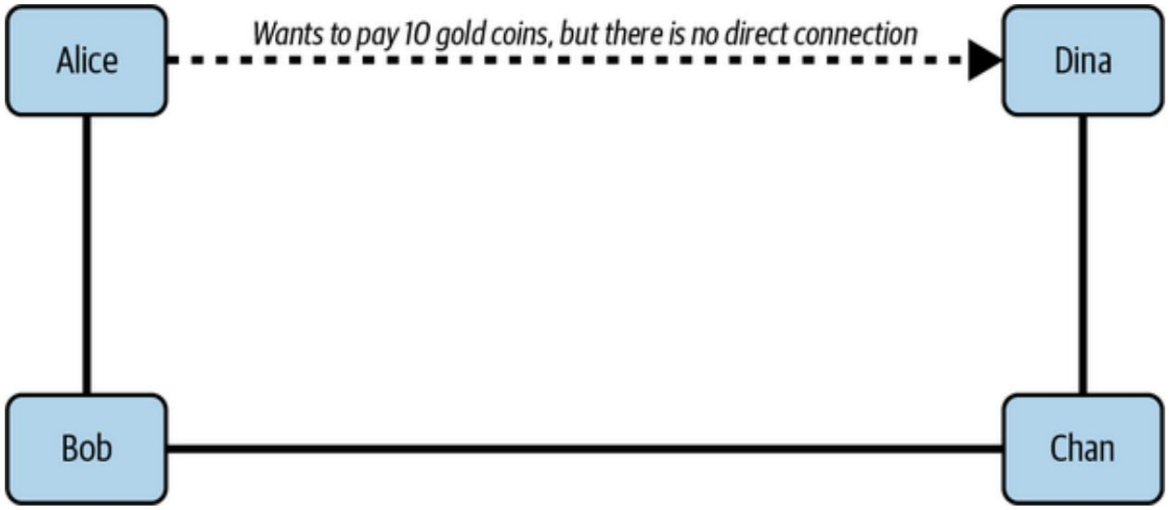

Figura 8-4. Alicia quiere pagarle a Dina 10 monedas de oro.

Alice puede pagarle a Bob para que le pague a Chan para que le pague a Dina, pero ¿cómo se asegura de que Bob o Chan no se escapen con las monedas después de recibirlas? En el mundo físico, los contratos podrían utilizarse para realizar de forma segura una serie de pagos.

Alice podría negociar un contrato con Bob, que dice:

> ***Yo, Alice, te daré, Bob, 10 monedas de oro si se las pasas a Chan.***

Si bien este contrato es agradable en abstracto, en el mundo real, Alice corre el riesgo de que Bob incumpla el contrato y espere que no lo atrapen. Incluso si atrapan a Bob y lo procesan, Alice corre el riesgo de que esté en bancarrota y no pueda devolverle sus 10 monedas de oro. Suponiendo que estos problemas se resuelvan mágicamente, aún no está claro cómo aprovechar dicho contrato para lograr el resultado deseado: entregar las monedas a Dina.

Mejoremos nuestro contrato para incorporar estas consideraciones:

> ***Yo, Alice, te reembolsaré, Bob, con 10 monedas de oro si puedes demostrarme (por ejemplo, a través de un recibo) que le has entregado 10 monedas de oro a Chan.***

Podría preguntarse por qué Bob debería firmar un contrato de este tipo. Tiene que pagarle a Chan, pero al final no obtiene nada del intercambio y corre el riesgo de que Alice no se lo reembolse. Bob podría ofrecerle a Chan un contrato similar para pagarle a Dina, pero Chan tampoco tiene motivos para aceptarlo.

Incluso dejando de lado el riesgo, Bob y Chan ***ya*** deben tener 10 monedas de oro para enviar; de lo contrario, no podrían participar en el contrato.

Por lo tanto, Bob y Chan enfrentan tanto el riesgo como el costo de oportunidad por aceptar este contrato, y necesitarían ser compensados para aceptarlo.

Entonces, Alice puede hacer que esto sea atractivo tanto para Bob como para Chan ofreciéndoles una moneda de oro a cada uno, si transmiten su pago a Dina.

Entonces el contrato diría:

> ***Yo, Alice, te reembolsaré, Bob, con 12 monedas de oro si puedes demostrarme (por ejemplo, a través de un recibo) que le has entregado 11 monedas de oro a Chan.***

Alice ahora le promete a Bob 12 monedas de oro. Hay 10 para entregar a Dina y 2 para los honorarios. Ella le promete 12 a Bob si él puede demostrar que ha enviado 11 a Chan. La diferencia de una moneda de oro es la tarifa que Bob ganará por ayudar con este pago en particular. En la Figura 8-5 vemos cómo este arreglo le daría 10 monedas de oro a Dina a través de Bob y Chan.

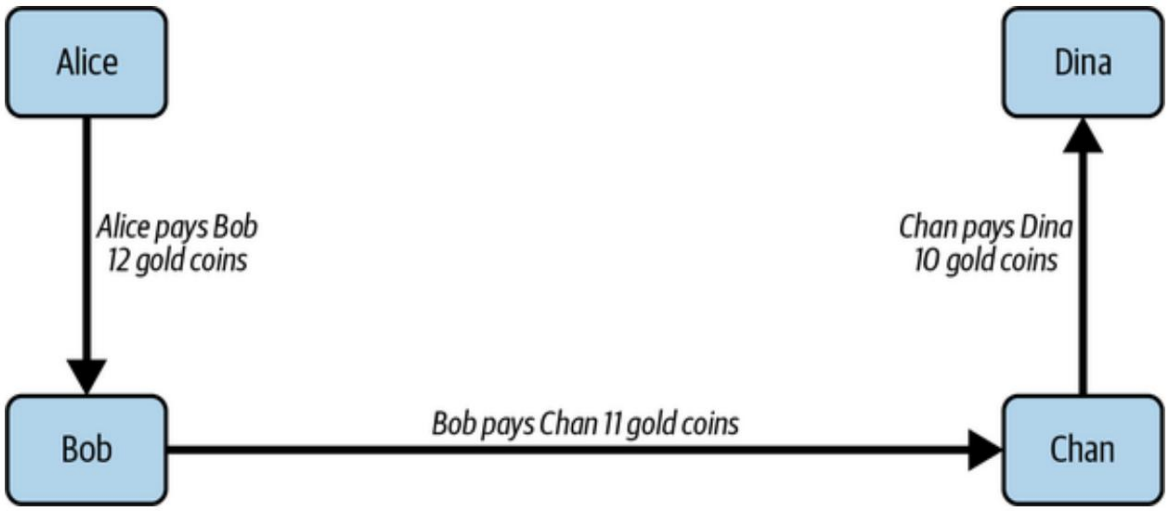

Figura 8-5. Alice país Bob, Bob país Chan, Chan país Dina

Debido a que todavía existe el problema de la confianza y el riesgo de que Alice o Bob no respeten el contrato, todas las partes deciden utilizar un servicio de depósito en garantía. Al comienzo del intercambio, Alice podría "bloquear" estas 12 monedas de oro en depósito en garantía que solo se le pagarán a Bob una vez que demuestre que le pagó 11 monedas de oro a Chan.

Este servicio de depósito en garantía es idealizado y no presenta otros riesgos (por ejemplo, riesgo de contraparte). Más adelante veremos cómo podemos reemplazar el depósito en garantía con un contrato inteligente de Bitcoin. Supongamos por ahora que todo el mundo confía en este servicio de depósito en garantía.

En Lightning Network, el recibo (comprobante de pago) podría tomar la forma de un secreto que solo Dina conoce. En la práctica, este secreto sería un número aleatorio lo suficientemente grande como para evitar que otros lo adivinen (¡típicamente un número ***muy, muy*** grande, codificado usando 256 bits!).

Dina genera este valor secreto R a partir de un generador de números aleatorios.

Luego, el secreto podría comprometerse con el contrato al incluir el hash SHA 256 del secreto en el contrato mismo, de la siguiente manera:

H = SHA-256 (R)

A este hash del secreto del ***pago lo llamamos hash de pago.*** El secreto que “desbloquea” el pago se llama ***secreto de pago.***

Por ahora, mantenemos las cosas simples y asumimos que el secreto de Dina es simplemente la línea de texto: secreto de Dina. Este mensaje secreto se denomina ***secreto*** de pago o ***preimagen de pago.***

Para "comprometerse" con este secreto, Dina calcula el hash SHA-256, que cuando se codifica en hexadecimal, se puede mostrar de la siguiente manera:

0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3

Para facilitar el pago de Alice, Dina creará el secreto de pago y el hash de pago, y enviará el hash de pago a Alice. En la Figura 8-6 vemos que Dina envía el hash de pago a Alice a través de algún canal externo (línea discontinua), como un correo electrónico o un mensaje de texto.

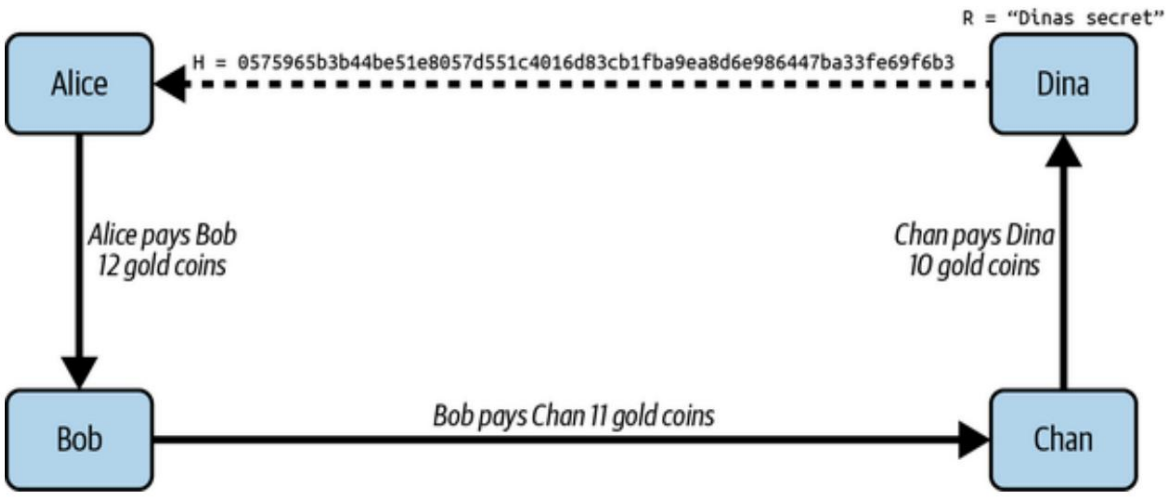

Figura 8-6. Dina envía el secreto hash a Alice

Alice no conoce el secreto, pero puede reescribir su contrato para usar el hash del secreto como prueba de pago:

> ***Yo, Alice, te reembolsaré, Bob, con 12 monedas de oro si puedes mostrarme un mensaje válido cuyo hash*** *sea: 057596….* ***Puede adquirir este mensaje estableciendo un contrato similar con Chan, quien tiene que establecer un contrato similar con Dina. Para asegurarle que se le reembolsará, proporcionaré las 12 monedas de oro a un fideicomiso de confianza antes de que establezca su próxima contrato.***

Este nuevo contrato ahora protege a Alice de que Bob no lo reenvíe a Chan, protege a Bob de que Alice no le reembolse y asegura que habrá pruebas de que a Dina finalmente se le pagó a través del hash del secreto de Dina.

Después de que Bob y Alice aceptan el contrato, y Bob recibe el mensaje del fideicomiso de que Alice ha depositado las 12 monedas de oro, Bob ahora puede negociar un contrato similar con Chan.

Tenga en cuenta que, dado que Bob cobra una tarifa de servicio de 1 moneda, solo enviará 11 monedas de oro a Chan una vez que Chan demuestre que le pagó a Dina. De manera similar, Chan también exigirá una tarifa y esperará recibir 11 monedas de oro una vez que haya demostrado que le pagó a Dina las 10 monedas de oro prometidas.

El contrato de Bob con Chan dirá:

> ***Yo, Bob, te reembolsaré, Chan, con 11 monedas de oro si puedes mostrarme un mensaje válido cuyo hash*** *sea: 057596….* ***Puede adquirir este mensaje estableciendo un contrato similar con Dina. Para asegurarle que se le reembolsará, proporcionaré las 11 monedas de oro a un fideicomiso de confianza antes de que establezca su próximo contrato.***

Una vez que Chan recibe el mensaje del fideicomiso de que Bob ha depositado las 11 monedas de oro, Chan establece un contrato similar con Dina:

> ***Yo, Chan, te reembolsaré a ti, Dina, con 10 monedas de oro si puedes mostrarme un mensaje válido cuyo hash*** *sea: 057596….* ***Para asegurarle que se le reembolsará después de revelar el secreto, proporcionaré las 10 monedas de oro a un fideicomiso de confianza.***

Todo está ahora en su lugar. Alice tiene un contrato con Bob y ha depositado 12 monedas de oro en depósito. Bob tiene un contrato con Chan y ha depositado 11 monedas de oro en depósito. Chan tiene un contrato con Dina y ha depositado 10 monedas de oro en depósito. Ahora le toca a Dina revelar el secreto, que es la preimagen del hash que ha establecido como prueba de pago.

Dina ahora le envía el secreto de Dina a Chan.

Chan comprueba que el hash secreto de Dina es 057596…. Chan ahora tiene

comprobante de pago y así instruye al servicio de depósito en garantía para que entregue las 10 monedas de oro a Dina.

Chan ahora le proporciona el secreto a Bob. Bob lo revisa e indica al servicio de depósito en garantía que entregue las 11 monedas de oro a Chan.

Bob ahora le proporciona el secreto a Alice. Alice lo revisa e indica al depósito en garantía que entregue 12 monedas de oro a Bob.

Todos los contratos ya están liquidados. Alice ha pagado un total de 12 monedas de oro, 1 de las cuales fue recibida por Bob, 1 de las cuales fue recibida por Chan y 10 de las cuales fueron recibidas por Dina. Con una cadena de contratos como este, Bob y Chan no pudieron huir con el dinero porque primero lo depositaron en depósito en garantía.

Sin embargo, aún queda un problema. Si Dina se negaba a revelar su preimagen secreta, entonces Chan, Bob y Alice tendrían sus monedas atascadas en el depósito en garantía, pero no serían reembolsadas. Y de manera similar, si alguien más a lo largo de la cadena no transmitiera el secreto, sucedería lo mismo. Entonces, si bien nadie puede robarle dinero a Alice, todos aún tendrían su dinero en depósito en garantía de forma permanente.

Afortunadamente, esto se puede resolver agregando una fecha límite al contrato.

Podríamos modificar el contrato para que, si no se cumple en un plazo determinado, el contrato caduque y el servicio de depósito en garantía devuelva el dinero a la persona que realizó el depósito original. A esta fecha límite la ***llamamos timelock.***

El depósito se bloquea con el servicio de depósito en garantía durante un cierto período de tiempo y finalmente se libera incluso si no se proporcionó un comprobante de pago.

Para tener esto en cuenta, el contrato entre Alice y Bob se modifica una vez más con una nueva cláusula:

> ***Bob tiene 24 horas para mostrar el secreto después de la firma del contrato. Si Bob no proporciona el secreto en ese momento, el depósito de Alice será reembolsado por el servicio de depósito en garantía y el contrato quedará invalidado.***

Bob, por supuesto, ahora debe asegurarse de recibir el comprobante de pago dentro de las 24 horas. Incluso si paga con éxito a Chan, si recibe el comprobante de pago más tarde de 24 horas, no se le reembolsará. Para eliminar ese riesgo, Bob debe darle a Chan un plazo aún más corto.

A su vez, Bob modificará su contrato con Chan de la siguiente manera:

> ***Chan tiene 22 horas para mostrar el secreto después de la firma del contrato. Si no proporciona el secreto en ese momento, el depósito de Bob será reembolsado por el servicio de depósito en garantía y el contrato quedará invalidado.***

Como habrás adivinado, Chan también alterará su contrato con Dina:

> ***Dina tiene 20 horas para mostrar el secreto después de la firma del contrato. Si ella no proporciona el secreto en ese momento, el depósito de Chan será reembolsado por el servicio de depósito en garantía y el contrato quedará invalidado.***

Con tal cadena de contratos podemos asegurar que, después de 24 horas, el pago pasará con éxito de Alice a Bob a Chan a Dina, o fallará y todos serán reembolsados. O el contrato fracasa o tiene éxito, no hay término medio.

En el contexto de Lightning Network, llamamos a esta propiedad de "todo o nada" ***atomicidad.***

Siempre que el depósito en garantía sea confiable y cumpla fielmente con su deber, no se robarán las monedas de ninguna de las partes en el proceso.

La condición previa para que esta ***ruta*** funcione es que todas las partes en la ruta tengan suficiente dinero para satisfacer la serie requerida de depósitos.

Si bien esto parece un detalle menor, veremos más adelante en este capítulo que este requisito es en realidad uno de los problemas más difíciles para los nodos LN. Se vuelve progresivamente más difícil a medida que aumenta el tamaño del pago.
Además, las partes no pueden usar su dinero mientras esté bloqueado depósito.

Por lo tanto, los usuarios que reenvían pagos enfrentan un costo de oportunidad por bloquear el dinero, que finalmente se reembolsa a través de tarifas de enrutamiento, como vimos en el ejemplo anterior.

Ahora que hemos visto un ejemplo de enrutamiento de pago físico, veremos cómo se puede implementar esto en la cadena de bloques de Bitcoin, sin necesidad de un depósito en garantía de terceros. Para hacer esto, configuraremos los contratos entre los participantes utilizando Bitcoin Script. Reemplazamos el depósito en garantía de terceros con ***contratos inteligentes*** que implementan un protocolo de equidad. ¡Desglosemos ese concepto e implementémoslo!

Protocolo de equidad

Como vimos en el primer capítulo de este libro, la innovación de Bitcoin es la capacidad de utilizar primitivas criptográficas para implementar un protocolo de equidad que sustituye la confianza en terceros (intermediarios) por un protocolo de confianza.

En nuestro ejemplo de la moneda de oro, necesitábamos un servicio de depósito en garantía para evitar que cualquiera de las partes incumpliera sus obligaciones. La innovación de los protocolos de equidad criptográfica nos permite reemplazar el servicio de custodia con un protocolo.

Las propiedades del protocolo de equidad que queremos crear son:

Operación sin confianza

Los participantes en un pago enrutado no necesitan confiar entre sí, ni en ningún intermediario o tercero. En cambio, confían en el protocolo para protegerlos de las trampas.

Atomicidad

O el pago se ejecuta por completo o falla y se reembolsa a todos. No existe la posibilidad de que un intermediario cobre un pago enrutado y no lo reenvíe al siguiente salto. Así, los intermediarios no pueden engañar ni robar.

multisalto

La seguridad del sistema se extiende de extremo a extremo para los pagos enrutados a través de múltiples canales de pago, al igual que para un pago entre los dos extremos de un solo canal de pago.

Una propiedad adicional opcional es la capacidad de dividir los pagos en varias partes mientras se mantiene la atomicidad de todo el pago. Estos se denominan ***pagos*** de varias partes ***(MPP)*** y se analizan con más detalle en "Pagos de varias partes".

Implementación de pagos Atomic Trustless Multihop

Bitcoin Script es lo suficientemente flexible como para que haya docenas de formas de implementar un protocolo de equidad que tenga las propiedades de atomicidad, operación sin confianza y seguridad multisalto. La elección de una implementación específica depende de ciertos compromisos entre privacidad, eficiencia y complejidad.

El protocolo de equidad para el enrutamiento utilizado en Lightning Network en la actualidad se denomina contrato de bloqueo de tiempo de hash (HTLC). Los HTLC usan una preimagen hash como el secreto que desbloquea un pago, como vimos en el ejemplo de la moneda de oro en este capítulo. El destinatario de un pago genera un número secreto aleatorio y calcula su hash. El hash se convierte en la condición de pago y, una vez que se revela el secreto, todos los participantes pueden canjear sus pagos entrantes. Los HTLC ofrecen atomicidad, funcionamiento sin confianza y seguridad multisalto.

Otro mecanismo propuesto para implementar el enrutamiento es un contrato de bloqueo de ***tiempo de punto (PTLC).*** Los PTLC también logran atomicidad, operación confiable y seguridad multisalto, pero lo hacen con mayor eficiencia y mejor privacidad. La implementación eficiente de los PTLC depende de un nuevo algoritmo de firma digital llamado ***firmas Schnorr,*** que se espera que se active en Bitcoin en 2021.

Revisando el ejemplo de las propinas

Repasemos nuestro ejemplo de la primera parte de este capítulo. Alice quiere darle una propina a Dina con un pago Lightning. Digamos que Alice quiere enviarle a Dina 50,000 satoshis como propina.

Para que Alice pague a Dina, Alice necesitará el nodo de Dina para generar una factura Lightning. Discutiremos esto con más detalle en el Capítulo 15. Por ahora, supongamos que Dina tiene un sitio web que puede generar una factura Lightning para propinas.

PROPINA

Los pagos relámpago se pueden enviar sin una factura utilizando una función llamada ***envío de claves,*** que analizaremos con más detalle en "Pagos espontáneos de envío de claves". Por ahora, explicaremos el flujo de pago más simple usando una factura.

Alice visita el sitio de Dina, ingresa la cantidad de 50 000 satoshis en un formulario y, en respuesta, el nodo Lightning de Dina genera una solicitud de pago por 50 000

satoshis en forma de factura Lightning. Esta interacción tiene lugar en la web y fuera de Lightning Network, como se muestra en la Figura 8-7.

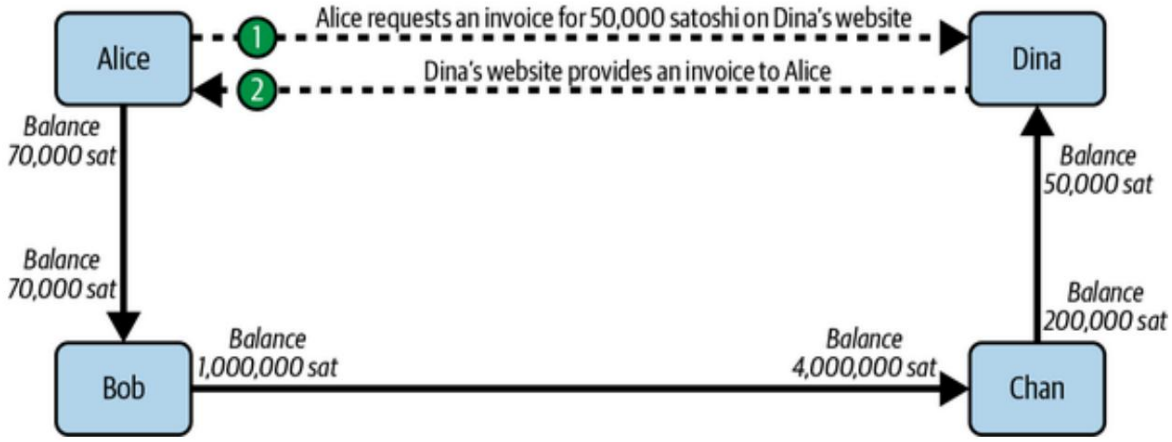

Figura 8-7. Alice solicita una factura del sitio web de Dina

Como vimos en ejemplos anteriores, asumimos que Alice no tiene un canal de pago directo a Dina. En cambio, Alice tiene un canal con Bob, Bob tiene un canal con Chan y Chan tiene un canal con Dina. Para pagarle a Dina, Alice debe encontrar un camino que la conecte con Dina. Discutiremos ese paso con más detalle en el Capítulo 12. Por ahora, supongamos que Alice puede recopilar información sobre los canales disponibles y ve que hay un camino desde ella hasta Dina, a través de Bob y Chan.

> **NOTA**
>
> ¿Recuerda cómo Bob y Chan podrían esperar una pequeña compensación por enrutar el pago a través de sus nodos? Alice quiere pagarle a Dina 50 000 satoshis, pero como verá en las siguientes secciones, le enviará a Bob 50 200 satoshis. Los 200 satoshis adicionales le pagarán a Bob y Chan 100 satoshis cada uno, como tarifa de enrutamiento.

Ahora, el nodo de Alice puede construir un pago Lightning. En las próximas secciones, veremos cómo el nodo de Alice construye un HTLC para pagar a Dina y cómo ese HTLC se reenvía a lo largo del camino de Alice a Dina.

Liquidación en cadena versus liquidación fuera de cadena de HTLC

El propósito de Lightning Network es permitir transacciones ***fuera de la cadena*** en las que se confíe de la misma manera que las transacciones en la cadena porque nadie puede hacer trampa. La razón por la que nadie puede hacer trampa es porque en cualquier momento, cualquiera de los participantes puede realizar sus transacciones fuera de la cadena en la cadena. Cada transacción fuera de la cadena está lista para enviarse a la cadena de bloques de Bitcoin en cualquier momento. Por lo tanto, la cadena de bloques de Bitcoin actúa como un mecanismo de resolución de disputas y solución final si es necesario.

El mero hecho de que cualquier transacción pueda realizarse en la cadena en cualquier momento es precisamente la razón por la que todas esas transacciones pueden mantenerse fuera de la cadena. Si sabe que tiene recurso, puede continuar cooperando con los otros participantes y evitar la necesidad de liquidación en cadena y tarifas adicionales.

En todos los ejemplos que siguen, supondremos que cualquiera de estas transacciones se puede realizar en cadena en cualquier momento. Los participantes optarán por mantenerlos fuera de la cadena, pero no hay diferencia en la funcionalidad del sistema aparte de las tarifas más altas y el retraso impuesto por la minería en cadena de las transacciones. El ejemplo funciona igual si todas las transacciones están dentro o fuera de la cadena.

Contratos bloqueados en el tiempo de hash

En esta sección explicamos cómo funcionan los HTLC.

La primera parte de un HTLC es el ***hash.*** Esto se refiere al uso de un algoritmo hash criptográfico para comprometerse con un secreto generado aleatoriamente.
El conocimiento del secreto permite la redención del pago. La función hash criptográfica garantiza que, si bien es imposible que alguien adivine la preimagen secreta, es fácil para cualquiera verificar el hash, y solo hay una preimagen posible que resuelve la condición de pago.

En la Figura 8-8 vemos a Alice recibiendo una factura Lightning de Dina. Dentro de esa factura, Dina ha codificado un ***hash de pago,*** que es el hash criptográfico de un secreto que produjo el nodo de Dina. El secreto de Dina se llama ***preimagen de pago.*** El hash de pago actúa como un identificador que se puede utilizar

para enviar el pago a Dina. La preimagen de pago actúa como recibo y comprobante de pago una vez que se completa el pago.

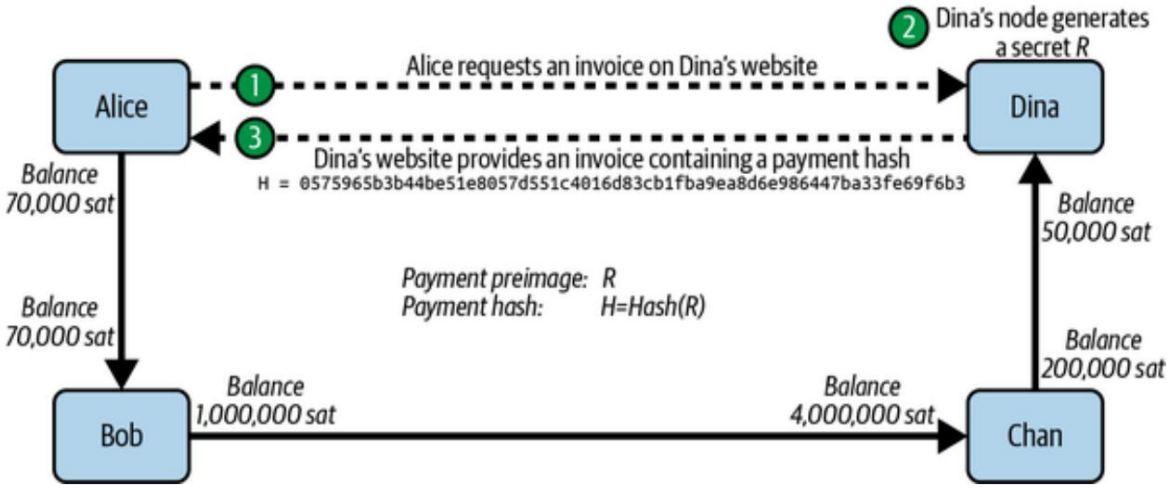

Figura 8-8. Alice recibe un hash de pago de Dina

En Lightning Network, la preimagen de pago de Dina no será una frase como el secreto de Dina, sino un número aleatorio generado por el nodo de Dina. Llamemos a ese número aleatorio R.

El nodo de Dina calculará un hash criptográfico de R, tal que:

$$H = \text{SHA-256}(R)$$

En esta ecuación, ***H*** es el hash o ***hash de pago*** y ***R*** es el secreto o ***preimagen de pago.***

El uso de una función hash criptográfica es un elemento que garantiza ***un funcionamiento sin confianza.*** Los intermediarios de pago no necesitan confiar unos en otros porque saben que nadie puede adivinar el secreto o falsificarlo.

HTLC en Bitcoin Script

En nuestro ejemplo de la moneda de oro, Alice tenía un contrato que se hizo cumplir mediante un depósito en garantía como este:

Alice le reembolsará a Bob 12 monedas de oro si puede mostrar un mensaje válido cuyo hash sea: *0575…f6b3.* ***Bob tiene 24 horas para mostrar el secreto después de la firma del contrato. Si Bob no proporciona el secreto en ese momento, el depósito de Alice será reembolsado por el servicio de depósito en garantía y el contrato quedará invalidado.***

Veamos cómo implementaríamos esto como un HTLC en Bitcoin Script. En el Ejemplo 8-1 , vemos un script de Bitcoin HTLC como se usa actualmente en Lightning Network. Puede encontrar esta definición en BOLT #3, Transacciones.

Ejemplo 8-1. HTLC implementado en Bitcoin Script (BOLT #3)

```
# Al nodo remoto con clave de revocación OP_DUP
OP_HASH160 <RIPEMD160(SHA256(recallpubkey))> OP_EQUAL OP_IF OP_CHECKSIG
OP_ELSE <remote_htlcpubkey> OP_SWAP OP_SIZE 32 OP_EQUAL OP_IF # Al nodo
    local a través de la transacción HTLC-success.

        OP_HASH160 <RIPEMD160(pago_hash)> OP_EQUALVERIFY 2
        OP_SWAP <local_htlcpubkey> 2 OP_CHECKMULTISIG OP_ELSE # Al
    nodo remoto después del tiempo de espera.

        OP_DROP <cltv_expiry> OP_CHECKLOCKTIMEVERIFY OP_DROP
        OP_CHECKSIG OP_ENDIF OP_ENDIF
```

¡Vaya, eso parece complicado! Sin embargo, no se preocupe, lo daremos paso a paso y lo simplificaremos.

El Bitcoin Script que se usa actualmente en Lightning Network es bastante complejo porque está optimizado para la eficiencia del espacio en la cadena, lo que lo hace muy compacto pero difícil de leer.

En las siguientes secciones, nos centraremos en los elementos principales de la secuencia de comandos y presentaremos secuencias de comandos simplificadas que son ligeramente diferentes de las que se usan realmente en Lightning.

La parte principal del HTLC está en la línea 10 del Ejemplo 8-1. ¡Vamos a construirlo desde cero!

Preimagen de pago y verificación de hash

El núcleo de un HTLC es el hash, donde se puede realizar el pago si el destinatario conoce la preimagen del pago. Alice bloquea el pago en un hash de pago específico y Bob tiene que presentar una preimagen de pago para reclamar los fondos. El sistema de Bitcoin puede verificar que la preimagen de pago de Bob sea correcta mediante un hash y comparando el resultado con el hash de pago que Alice usó para bloquear los fondos.

Esta parte de un HTLC se puede implementar en Bitcoin Script de la siguiente manera:

```
OP_SHA256 <H> OP_EQUAL
```

Alice puede crear una salida de transacción que pague, 50,200 satoshi con un script de bloqueo arriba, reemplazando <H> con el valor hash 0575…f6b3 proporcionado por Dina. Luego, Alice puede firmar esta transacción y ofrecérsela a Bob:

```
OP_SHA256 0575...f6b3 OP_EQUAL
```

Bob no puede gastar este HTLC hasta que sepa el secreto de Dina, por lo que gastar el HTLC está condicionado a que Bob cumpla con el pago hasta Dina.

Una vez que Bob tiene el secreto de Dina, Bob puede gastar esta salida con un script de desbloqueo que contiene el valor de preimagen secreto R.

El script de desbloqueo combinado con el script de bloqueo produciría:

```
<R> OP_SHA256 <H> OP_EQUAL
```

El motor Bitcoin Script evaluaría este script de la siguiente manera:

1. R se empuja a la pila.
2. El operador OP_SHA256 toma el valor R de la pila y lo procesa, empujando el resultado H_R a la pila.
3. H es empujado hacia la pila.

4. El operador OP_EQUAL compara H y el resultado $_R$. Si son iguales, el H es VERDADERO, el script está completo y el pago está verificado.

Ampliación de los HTLC de Alice a Dina

Alice ahora extenderá el HTLC a través de la red para que llegue a Dina.

En la Figura 8-9, vemos que el HTLC se propaga a través de la red desde Alice hasta Dina. Alice le ha dado a Bob un HTLC por 50 200 satoshi. Bob ahora puede crear un HTLC por 50,100 satoshi y dárselo a Chan.

Bob sabe que Chan no puede canjear el HTLC de Bob sin transmitir el secreto, momento en el que Bob también puede usar el secreto para canjear el HTLC de Alice. Este es un punto realmente importante porque asegura la ***atomicidad*** de extremo a extremo del HTLC. Para gastar el HTLC, uno debe revelar el secreto, lo que hace posible que otros también gasten su HTLC. O todos los HTLC son gastables, o ninguno de los HTLC es gastable: ¡atomicidad!

Debido a que el HTLC de Alice es 100 satoshi más que el HTLC que Bob le dio a Chan, Bob ganará 100 satoshi como tarifa de enrutamiento si este pago se completa.

Bob no se arriesga y no confía en Alice ni en Chan. En cambio, Bob confía en que una transacción firmada junto con el secreto se podrá canjear en la cadena de bloques de Bitcoin.

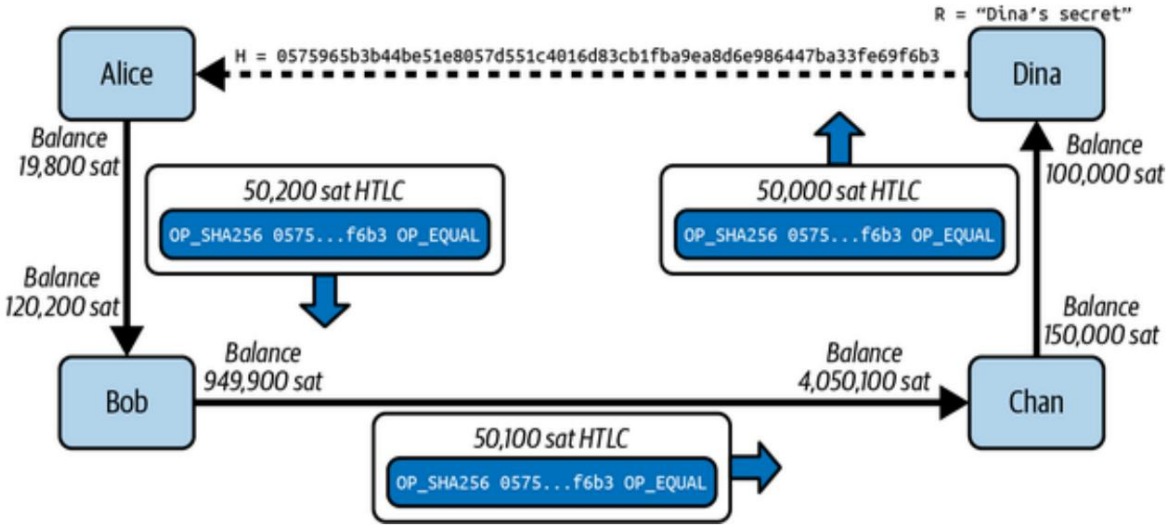

Figura 8-9. Propagación del HTLC a través de la red

De manera similar, Chan puede extender un HTLC de 50,000 a Dina. No está arriesgando nada ni confiando en Bob o Dina. Para canjear el HTLC, Dina tendría que transmitir el secreto, que Chan podría usar para canjear el HTLC de Bob. Chan también ganaría 100 satoshis como tarifa de enrutamiento.

Retropropagando el secreto

Una vez que Dina recibe un HTLC de 50,000 de Chan, ahora puede recibir el pago. Dina podría simplemente cometer este HTLC en cadena y gastarlo revelando el secreto en la transacción de gasto. O, en cambio, Dina puede actualizar el balance del canal con Chan al darle el secreto. No hay razón para incurrir en una tarifa de transacción y conectarse a la cadena. Entonces, en lugar de eso, Dina le envía el secreto a Chan, y acuerdan actualizar los saldos de sus canales para reflejar un pago Lightning de 50,000 satoshi a Dina. En la Figura 8-10 vemos a Dina dándole el secreto a Chan, cumpliendo así el HTLC.

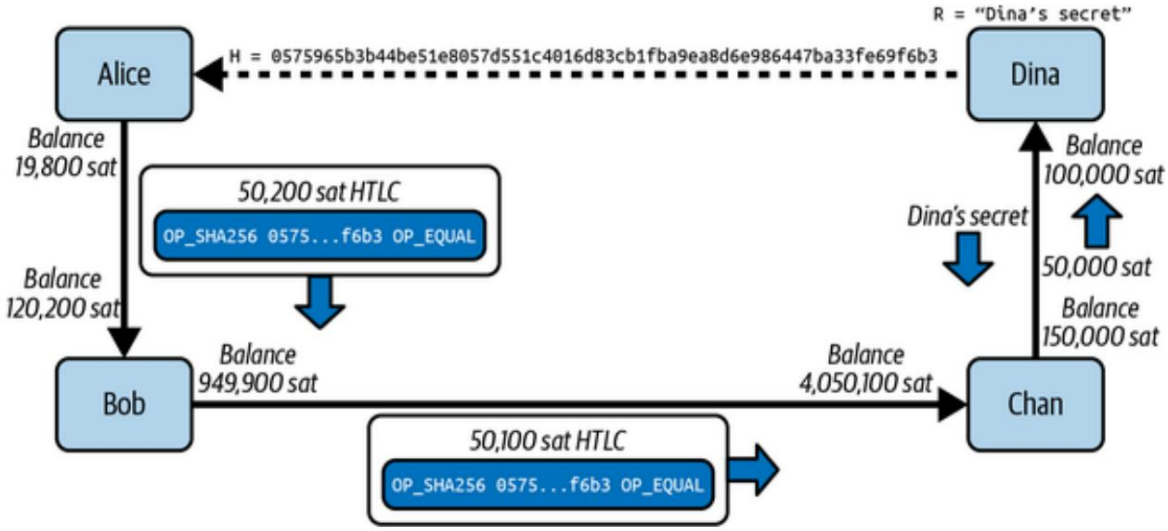

Figura 8-10. Dina liquida el HTLC de Chan fuera de la cadena

Observe que el saldo del canal de Dina va de 50 000 satoshi a 100 000 satoshi. El balance del canal de Chan se reduce de 200 000 satoshi a 150 000 satoshi. La capacidad del canal no ha cambiado, pero 50,000 se han movido del lado del canal de Chan al lado del canal de Dina.

Chan ahora tiene el secreto y le ha pagado a Dina 50,000 satoshi. Él puede hacer esto sin ningún riesgo, porque el secreto le permite a Chan canjear los 50,100

HTLC de Bob. Chan tiene la opción de comprometer ese HTLC en la cadena y gastarlo al revelar el secreto en la cadena de bloques de Bitcoin. Pero, al igual que Dina, prefiere evitar las tarifas de transacción. Entonces, en cambio, envía el secreto a Bob para que puedan actualizar los saldos de sus canales para reflejar un pago Lightning de 50,100 satoshi de Bob a Chan. En la Figura 8-11 vemos a Chan enviando el secreto a Bob y recibiendo un pago a cambio.

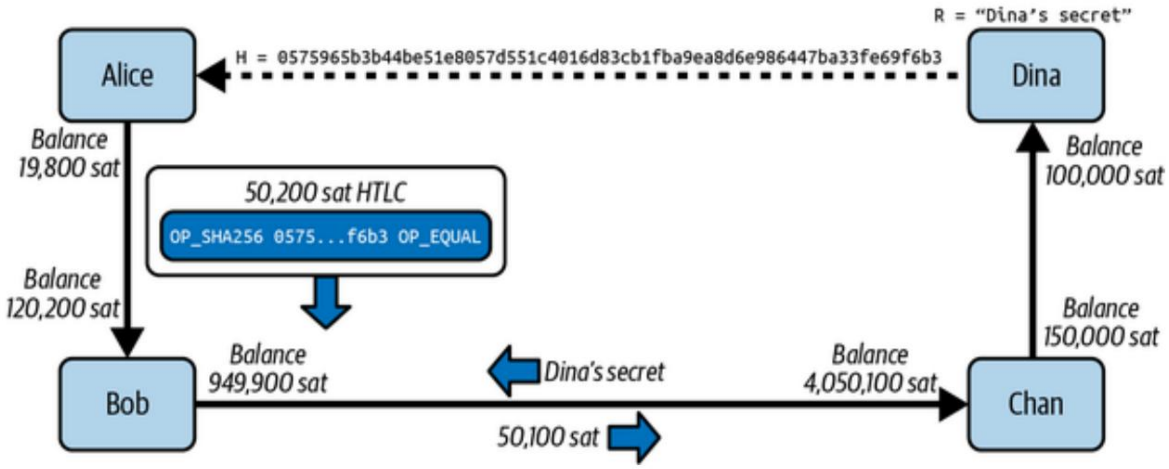

Figura 8-11. Chan liquida el HTLC de Bob fuera de la cadena

Chan le pagó a Dina 50 000 satoshi y recibió 50 100 satoshi de Bob.
Entonces Chan tiene 100 satoshi más en los saldos de su canal, que ganó como tarifa de enrutamiento.

Bob ahora también tiene el secreto. Puede usarlo para gastar el HTLC de Alice en cadena.
O bien, puede evitar las tarifas de transacción liquidando el HTLC en el canal con Alice.
En la Figura 8-12 vemos que Bob envía el secreto a Alice y ellos actualizan el saldo del canal para reflejar un pago Lightning de 50 200 satoshi de Alice a Bob.

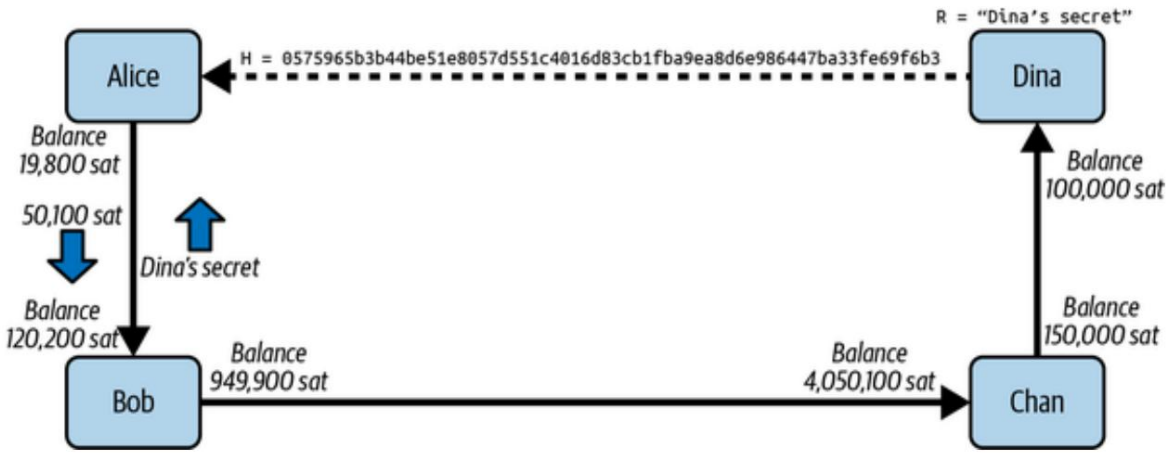

Figura 8-12. Bob liquida el HTLC de Alice fuera de la cadena

Bob recibió 50ÿ200 satoshi de Alice y le pagó 50ÿ100 satoshi a Chan, por lo que tiene 100 satoshi adicionales en los saldos de su canal debido a las tarifas de enrutamiento.

Alice recibe el secreto y ha liquidado los 50.200 satoshi HTLC. El secreto se puede usar como ***recibo*** para demostrar que a Dina se le pagó por ese hash de pago específico.

Los saldos finales del canal reflejan el pago de Alice a Dina y las tarifas de enrutamiento pagadas en cada salto, como se muestra en la Figura 8-13.

Figura 8-13. Saldos del canal después del pago

Encuadernación de firmas: prevención del robo de HTLC

Hay una trampa. ¿Lo notaste?

Si Alice, Bob y Chan crean los HTLC como se muestra en la Figura 8-13, enfrentan un riesgo de pérdida pequeño pero no insignificante. Cualquiera de esos HTLC puede ser canjeado (gastado) por cualquier persona que conozca el secreto. Al principio solo Dina conoce el secreto. Se supone que Dina solo debe gastar el HTLC de Chan. ¡Pero Dina podría gastar los tres HTLC al mismo tiempo, o incluso en una sola transacción de gasto! Después de todo, Dina conoce el secreto antes que nadie.
De manera similar, una vez que Chan conoce el secreto, se supone que solo debe gastar el HTLC ofrecido por Bob. Pero, ¿y si Chan también gasta el HTLC ofrecido por Alice?

¡ Esto no es ***desconfiado!*** Falla en la función de seguridad más importante. Necesitamos corregir esto.

El script HTLC debe tener una condición adicional que vincule cada HTLC a un destinatario específico. Hacemos esto al requerir una firma digital que coincida con la clave pública de cada destinatario, evitando así que alguien más gaste ese HTLC. Dado que solo el destinatario designado tiene la capacidad de producir una firma digital que coincida con esa clave pública, solo el destinatario designado puede gastar ese HTLC.

Miremos los scripts nuevamente con esta modificación en mente. El HTLC de Alice para Bob se modifica para incluir la clave pública de Bob y el operador OP_CHECKSIG.

Aquí está el script HTLC modificado:

```
OP_SHA256 <H> OP_EQUALVERIFY <Pub de Bob> OP_CHECKSIG
```

PROPINA

Tenga en cuenta que también cambiamos OP_EQUAL a OP_EQUALVERIFY. Cuando un operador tiene el sufijo VERIFICAR, no devuelve VERDADERO o FALSO en la pila. En cambio, ***detiene*** la ejecución y falla el script si el resultado es falso y continúa sin ningún resultado de pila si es verdadero.

Para canjear este HTLC, Bob debe presentar un script de desbloqueo que incluya una firma de la clave privada de Bob, así como la preimagen secreta de pago,

como esto:

<Firma de Bob> <R>

El motor de secuencias de comandos combina y evalúa los scripts de desbloqueo y bloqueo, de la siguiente manera:

<Firma de Bob> <R> OP_SHA256 <H> OP_EQUALVERIFY <Pub de Bob>
OP_CHECKSIG

1. <Bob's Sig> se empuja a la pila.
2. R se empuja a la pila.
3. OP_SHA256 extrae y hash R desde la parte superior de la pila y empuja H a la pila.
4. H es empujado hacia la pila.
5. OP_EQUALVERIFY extrae H y H y los compara. Si no son iguales, la ejecución se detiene. De lo contrario, continuamos sin salida a la pila.
6. La tecla <Bob's Pub> se empuja a la pila.
7. OP_CHECKSIG muestra <Bob's Sig> y <Bob's Pub> y verifica la firma. El resultado (VERDADERO/FALSO) se empuja a la pila.

Como puede ver, esto es un poco más complicado, pero ahora hemos arreglado el HTLC y nos hemos asegurado de que solo el destinatario previsto pueda gastarlo.

Optimización de hash

Veamos la primera parte del script HTLC hasta ahora:

OP_SHA256 <H> OP_EQUALVERIFICAR

Si observamos esto en la representación simbólica anterior, parece que los operadores OP_ ocupan la mayor parte del espacio. Pero ese no es el caso. Bitcoin Script está codificado en binario, y cada operador representa un byte. Mientras tanto, el valor <H> que usamos como marcador de posición para el hash de pago es un valor de 32 bytes (256 bits). Puede encontrar una lista de todos los operadores de Bitcoin Script y su codificación binaria y hexadecimal en Bitcoin Wiki: Script, o en el Apéndice D, "Operadores, constantes y símbolos del lenguaje de secuencias de comandos de transacciones", en ***Mastering Bitcoin.***

Representado en hexadecimal, nuestro script HTLC se vería así:

```
a8
0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3
88
```

En codificación hexadecimal, OP_SHA256 es a8 y OP_EQUALVERIFY es 88. La longitud total de este script es de 34 bytes, de los cuales 32 bytes son el hash.

Como mencionamos anteriormente, cualquier participante en Lightning Network debería poder tomar una transacción fuera de la cadena que tenga y ponerla en la cadena si necesita hacer cumplir su reclamo de fondos. Para realizar una transacción en cadena, tendrían que pagar tarifas de transacción a los mineros, y estas tarifas son proporcionales al tamaño, en bytes, de la transacción.

Por lo tanto, queremos encontrar formas de minimizar el "peso" de las transacciones en la cadena optimizando el script tanto como sea posible. Una forma de hacerlo es agregar otra función hash además del algoritmo SHA-256, una que produzca hashes más pequeños. El lenguaje Bitcoin Script proporciona el operador OP_HASH160 que "doble hash" de una preimagen: primero se aplica hash a la preimagen con SHA-256, y luego se aplica hash al hash resultante nuevamente con el algoritmo hash RIPEMD160. El hash resultante de RIPEMD160 es de 160 bits o 20 bytes, mucho más compacto. En Bitcoin Script, esta es una optimización muy común que se usa en muchos de los formatos de dirección comunes.

Entonces, usemos esa optimización en su lugar. Nuestro hash SHA-256 es 057596… 69f6b3. Poner eso a través de otra ronda de hashing con RIPEMD160

nos da el resultado:

```
R = "Servicio secreto"
H256 = SHA256(R)
H256 =
0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3
H160 = MADUROMD160(H256)
H160 = 9e017f6767971ed7cea17f98528d5f5c0ccb2c71
```

Alice puede calcular el hash RIPEMD160 del hash de pago que proporciona Dina y usar el hash más corto en su HTLC, ¡al igual que Bob y Chan!

El script HTLC "optimizado" se vería así:

```
OP_HASH160 <H160> OP_EQUALVERIFICAR
```

Codificado en hexadecimal, esto es:

```
a9 9e017f6767971ed7cea17f98528d5f5c0ccb2c71 88
```

Donde OP_HASH160 es a9 y OP_EQUALVERIFY es 88. ¡Este script tiene solo 22 bytes! Hemos ahorrado 12 bytes de cada transacción que canjea un HTLC en cadena.

Con esa optimización, ahora ve cómo llegamos al script HTLC que se muestra en la línea 10 del Ejemplo 8-1:

```
...
    # Al nodo local a través de la transacción HTLC-success.
    OP_HASH160 <RIPEMD160(pago_hash)> OP_EQUALVERIFY...
```

Fallo de tiempo de espera y cooperativo de HTLC

Hasta ahora, analizamos la parte "hash" de HTLC y cómo funcionaría si todos cooperaran y estuvieran en línea en el momento del pago.

¿Qué sucede si alguien se desconecta o no coopera? ¿Qué sucede si el pago no puede tener éxito?

Necesitamos asegurar una forma de "fallar con gracia", porque las fallas de enrutamiento ocasionales son inevitables. Hay dos formas de fallar: cooperativamente y con un reembolso de tiempo limitado.

La falla cooperativa es relativamente simple: cada participante en la ruta deshace el HTLC, eliminando el resultado del HTLC de sus transacciones de compromiso sin cambiar el saldo. Veremos cómo funciona eso en detalle en el Capítulo 9.

Veamos cómo podemos revertir un HTLC sin la cooperación de uno o más participantes. Necesitamos asegurarnos de que si uno de los participantes no coopera, los fondos no queden simplemente bloqueados en el HTLC ***para siempre.***
Esto le daría a alguien la oportunidad de rescatar los fondos de otro participante: “Dejaré sus fondos inmovilizados para siempre si no me paga el rescate”.

Para evitar esto, cada secuencia de comandos HTLC incluye una cláusula de reembolso que está conectada a un bloqueo de tiempo. ¿Recuerda nuestro contrato de depósito en garantía original? “Bob tiene 24 horas para mostrar el secreto después de la firma del contrato. Si Bob no proporciona el secreto en ese momento, se reembolsará el depósito de Alice”.

El reembolso con límite de tiempo es una parte importante del guión que garantiza la ***atomicidad, de*** modo que todo el pago de extremo a extremo tenga éxito o falle con gracia.
No hay un estado "medio pagado" del que preocuparse. Si hay una falla, cada participante puede desconectar el HTLC en cooperación con su socio de canal o poner la transacción de reembolso con bloqueo de tiempo en la cadena de manera unilateral para recuperar su dinero.

Para implementar este reembolso en Bitcoin Script, usamos un operador especial EN _ CHECKLOCKTIMEVERIFY también conocido como OP_CLTV para abreviar. Aquí está el guión, como se vio anteriormente en la línea 13 del Ejemplo 8-1:

```
...
        OP_DROP <cltv_expiry> OP_CHECKLOCKTIMEVERIFY OP_DROP
        OP_CHECKSIG
...
```

El operador OP_CLTV toma un tiempo de vencimiento definido como la altura del bloque después del cual esta transacción es válida. Si el bloqueo de tiempo de la transacción no está establecido, el igual que <cltv_expiry>, la evaluación del script falla y la transacción no es válida. De lo contrario, la secuencia de comandos continúa sin ninguna salida a la pila. Recuerde, el sufijo VERIFICAR significa que este operador no genera VERDADERO o FALSO, sino que se detiene/falla o continúa sin salida de pila.

Esencialmente, OP_CLTV actúa como un "guardián" que evita que el script continúe si no se ha alcanzado la altura del bloque <cltv_expiry> en la cadena de bloques de Bitcoin.

El operador OP_DROP simplemente coloca el elemento superior en la pila de scripts.
Esto es necesario al principio porque hay un elemento "sobrante" de las líneas de guión anteriores. Es necesario ***después*** de OP_CLTV eliminar el elemento <cltv_expiry> de la parte superior de la pila porque ya no es necesario.

Finalmente, una vez que se haya limpiado la pila, debe quedar una clave pública y una firma que OP_CHECKSIG pueda verificar. Como vimos en "Enlace de firma: prevención del robo de HTLC", esto es necesario para garantizar que solo el propietario legítimo de los fondos pueda reclamarlos, vinculando esta salida a su clave pública y requiriendo una firma.

Bloqueos de tiempo decrecientes

A medida que los HTLC se extienden de Alice a Dina, la cláusula de reembolso con límite de tiempo en cada HTLC tiene un valor cltv_expiry ***diferente .*** Veremos esto con más detalle en el Capítulo 10. Pero baste decir que para garantizar una cancelación ordenada de un pago que falla, cada salto debe esperar un poco menos por su reembolso. La diferencia entre los bloqueos de tiempo para cada salto se denomina cltv_expiry_delta, y la establece cada nodo y se anuncia a la red, como veremos en el Capítulo 11.

Por ejemplo, Alice establece el bloqueo de tiempo de reembolso en el primer HTLC a una altura de bloque de + 500 bloques actuales ("actual" es la altura de bloque actual).

Bob luego configuraría el bloqueo de tiempo cltv_expiry en el HTLC a Chan a + 450 bloques actuales. Chan establecería el bloqueo de tiempo a + 400 bloques actuales desde la altura del bloque actual. De esta forma, Chan puede obtener un reembolso por el HTLC que le ofreció a Dina ***antes*** de que Bob obtenga un reembolso por el HTLC que le ofreció a Chan. Bob puede obtener un reembolso del HTLC que le ofreció a Chan antes de que Alice pueda obtener un reembolso del HTLC que le ofreció a Bob. El bloqueo de tiempo decreciente evita las condiciones de carrera y garantiza que la cadena HTLC se desenrolle hacia atrás, desde el destino hacia el origen.

Conclusión

En este capítulo vimos cómo Alice puede pagar a Dina incluso si no tiene un canal de pago directo. Alice puede encontrar una ruta que la conecte con Dina y enrutar un pago a través de varios canales de pago para que llegue a Dina.

Para garantizar que el pago sea atómico y confiable en múltiples saltos, Alice debe implementar un protocolo de equidad en cooperación con todos los nodos intermediarios en la ruta. El protocolo de equidad se implementa actualmente como un HTLC, que asigna fondos a un hash de pago derivado de una preimagen de pago secreta.

Cada uno de los participantes en la ruta de pago puede extender un HTLC al siguiente participante, sin preocuparse por robos o fondos atascados. El HTLC se puede canjear revelando la preimagen de pago secreta. Una vez que un HTLC llega a Dina, ella revela la preimagen, que fluye hacia atrás, resolviendo todos los HTLC ofrecidos.

Finalmente, vimos cómo una cláusula de reembolso con límite de tiempo completa el HTLC, asegurando que cada participante pueda obtener un reembolso si el pago falla, pero por alguna razón uno de los participantes no coopera para deshacer los HTLC. Al tener siempre la opción de conectarse a la cadena para obtener un reembolso, el HTLC logra el objetivo de imparcialidad de atomicidad y operación sin confianza.

Capítulo 9. Operación del Canal y Reenvío de Pagos

En este capítulo, reuniremos los canales de pago y los contratos bloqueados en el tiempo de hash (HTLC). En el Capítulo 7, explicamos la forma en que Alice y Bob construyen un canal de pago entre sus dos nodos. También analizamos los mecanismos de compromiso y penalización que aseguran el canal de pago. En el Capítulo 8, analizamos los HTLC y cómo se pueden usar para enrutar un pago a través de una ruta formada por múltiples canales de pago. En este capítulo, reunimos los dos conceptos al observar cómo se administran los HTLC en cada canal de pago, cómo se comprometen los HTLC con el estado del canal y cómo se liquidan para actualizar los saldos del canal.

Específicamente, discutiremos "Agregar, establecer, fallar HTLC" y la "Máquina de estado del canal" que forman la superposición entre la capa de igual a igual y la capa de enrutamiento, como se destaca en un esquema en la Figura 9-1.

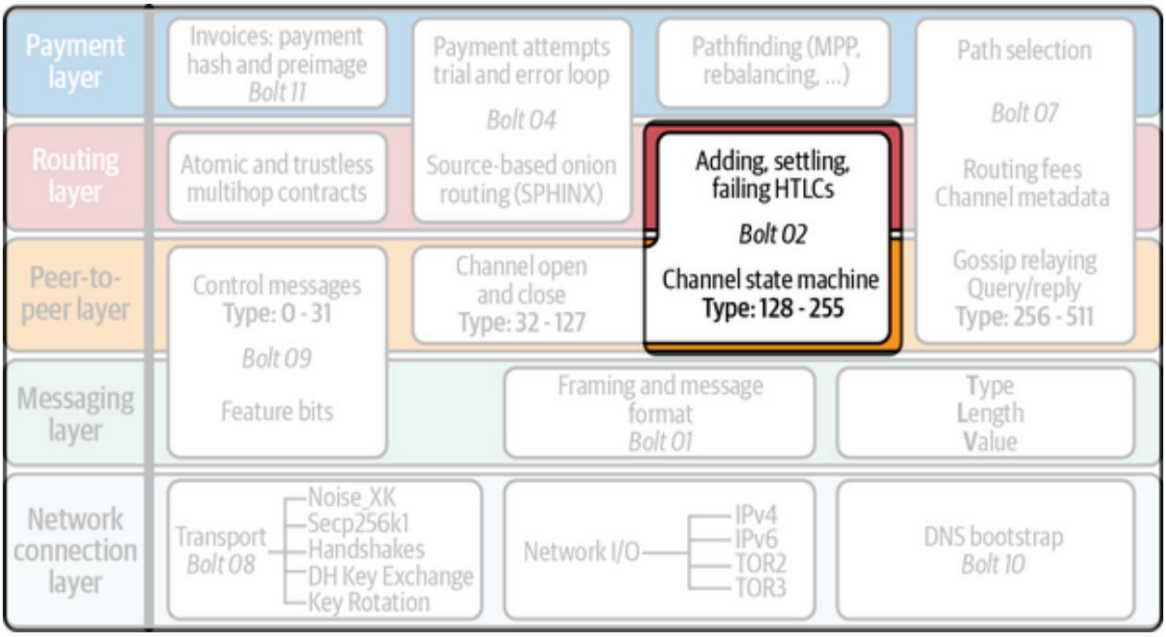

Figura 9-1. Operación de canales y reenvío de pagos en la suite de protocolos Lightning

Local (canal único) versus enrutado (Múltiples canales)

Aunque es posible enviar pagos a través de un canal de pago simplemente actualizando los saldos del canal y creando nuevas transacciones de compromiso, el protocolo Lightning usa HTLC incluso para pagos "locales" a través de un canal de pago. La razón de esto es mantener el mismo diseño de protocolo independientemente de si un pago es solo un salto (a través de un único canal de pago) o varios saltos (enrutado a través de múltiples canales de pago).

Al mantener la misma abstracción tanto para lo local como para lo remoto, no solo simplificamos el diseño del protocolo sino que también mejoramos la privacidad. Para el destinatario de un pago, no existe una diferencia perceptible entre un pago realizado directamente por su socio de canal y un pago reenviado por su socio de canal en nombre de otra persona.

Reenvío de pagos y actualización Compromisos con los HTLC

Revisaremos nuestro ejemplo del Capítulo 8 para demostrar cómo los HTLC de Alice a Dina se comprometen con cada canal de pago. Como recordará en nuestro ejemplo, Alice le paga a Dina 50 000 satoshis al enrutar un HTLC a través de Bob y Chan. La red se muestra en la Figura 9-2.

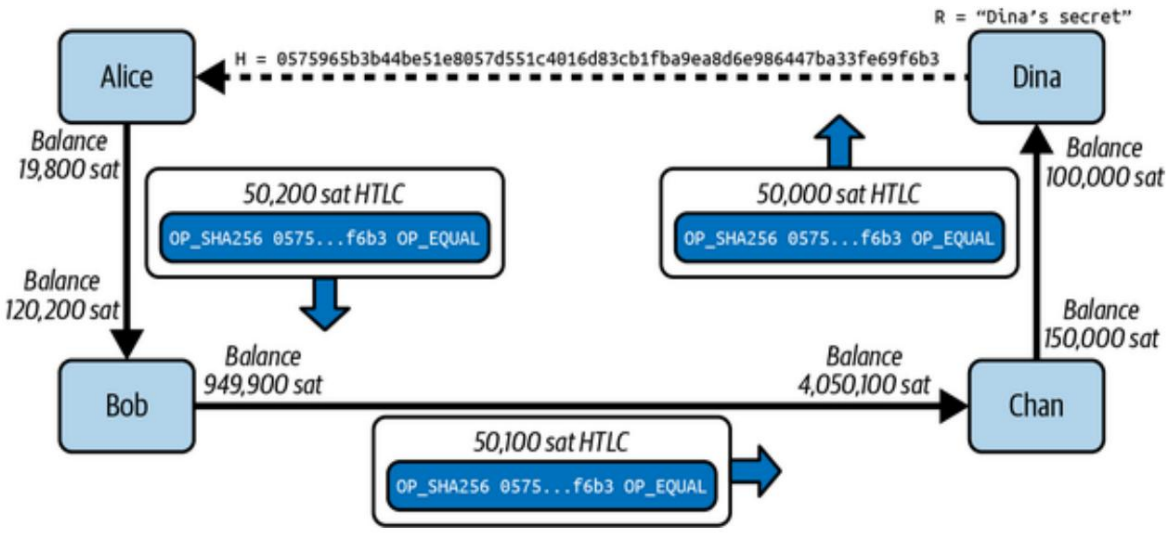

Figura 9-2. Alice le paga a Dina con un HTLC enrutado a través de Bob y Chan

Nos centraremos en el canal de pago entre Alice y Bob y revisaremos los mensajes y las transacciones que utilizan para procesar este HTLC.

HTLC y flujo de mensajes de compromiso

El flujo de mensajes entre Alice y Bob (y también entre cualquier par de socios de canal) se muestra en la figura 9-3.

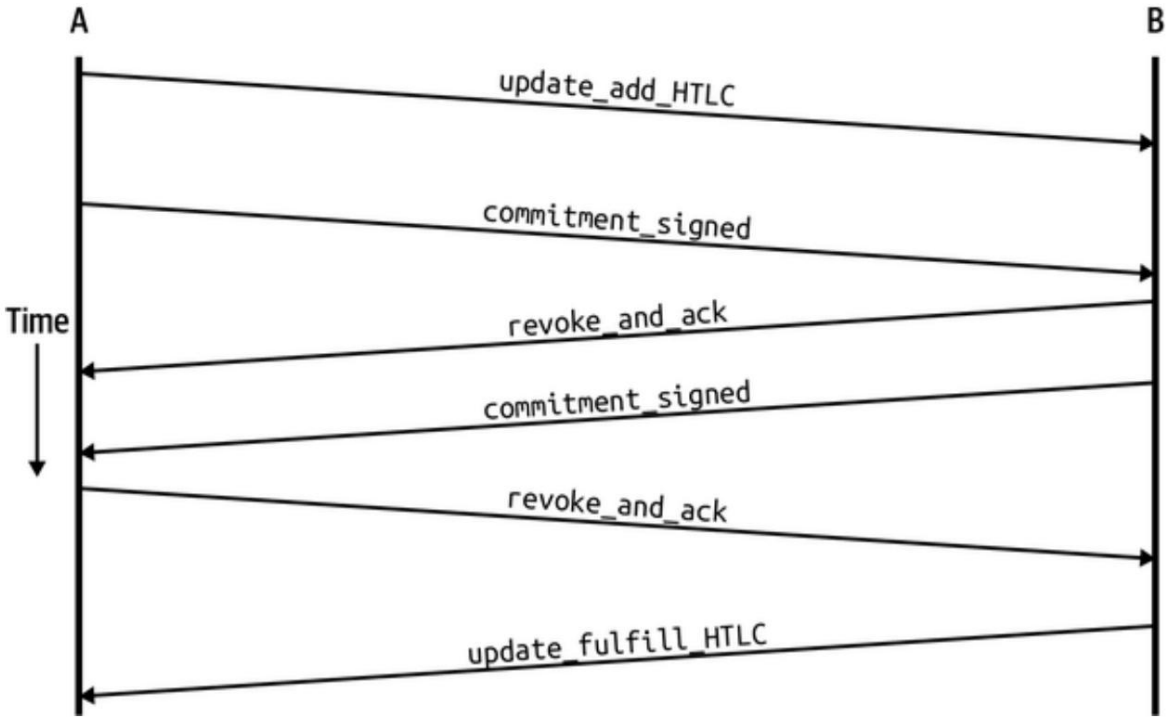

Figura 9-3. El flujo de mensajes para el compromiso de HTLC entre socios de canal

Ya vimos el compromiso_firmado y el revoque_y_ack en el Capítulo 7. Ahora veremos cómo encajan los HTLC en el esquema de compromiso. Los dos nuevos mensajes son update_add_htlc, que Alice usa para pedirle a Bob que agregue un HTLC, y update_fulfill_htlc, que Bob usa para canjear el HTLC una vez que ha recibido el secreto de pago (el secreto de Dina).

Reenvío de pagos con HTLC

Alice y Bob comienzan con un canal de pago que tiene un saldo de 70 000 satoshi en cada lado.

Como vimos en el Capítulo 7, esto significa que Alice y Bob han negociado y cada uno tiene transacciones de compromiso. Estas transacciones de compromiso son asimétricas, retrasadas y revocables, y se parecen al ejemplo de la figura 9-4 .

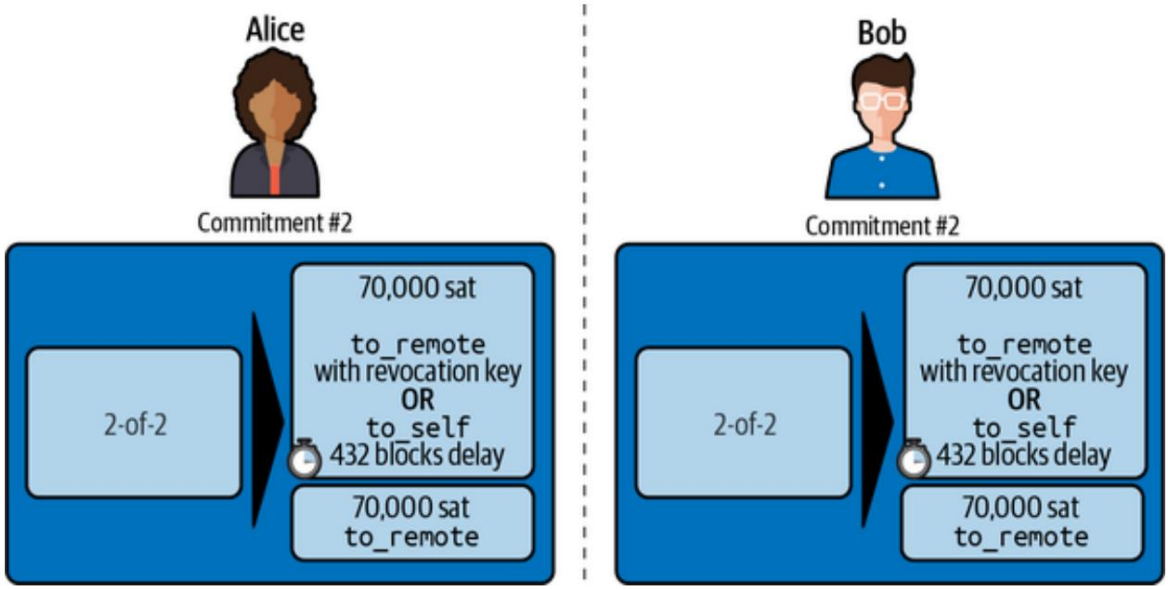

Figura 9-4. Transacciones de compromiso inicial de Alice y Bob

Agregar un HTLC

Alice quiere que Bob acepte un HTLC por valor de 50ÿ200 satoshis para enviárselo a Dina. Para hacerlo, Alice debe enviar los detalles de este HTLC, incluidos el hash y el monto del pago, a Bob. Bob también necesitará saber dónde reenviarlo, que es algo que discutiremos en detalle en el Capítulo 10.

Para agregar el HTLC, Alice inicia el flujo que vimos en la Figura 9-3 enviando el mensaje update_add_htlc a Bob.

El mensaje update_add_HTLC Alice envía

el mensaje update_add_HTLC Lightning a Bob. Este mensaje se define en BOLT #2: Peer Protocol, update_add_HTLC, y se muestra en el Ejemplo 9-1.

***Ejemplo 9-1. El mensaje** update_add_HTLC*

```
[channel_id:channel_id] [u64:id]
[u64:amount_msat]
[sha256:pago_hash] [u32:cltv_expiry]
[1366*byte:onion_routing_packet]
```

Canal ID

Este es el canal que Alice tiene con Bob donde quiere agregar el HTLC. Recuerde que Alice y Bob pueden tener múltiples canales entre sí.

identificación

Este es un contador de HTLC y comienza en 0 para el primer HTLC ofrecido a Bob por Alice y se incrementa para cada HTLC ofrecido posteriormente.

cantidad_msat

Esta es la cantidad (valor) del HTLC en millisatoshis. En nuestro ejemplo, esto es 50,200,000 millisatoshis (es decir, 50,200 satoshis).

pago_hash

Este es el hash de pago calculado a partir de la factura de Dina. Es ***H*** = RIPEMD160 (SHA-256 (R)), donde ***R*** es el secreto de Dina que solo Dina conoce y se revelará si se le paga a Dina.

cltv_expiry

Este es el tiempo de vencimiento de este HTLC, que se codificará como un reembolso de tiempo limitado en caso de que el HTLC no llegue a Dina en este tiempo.

paquete_enrutamiento_cebolla

Esta es una ruta encriptada con cebolla que le dice a Bob dónde reenviar este HTLC a continuación (a Chan). El enrutamiento de cebolla se cubre en detalle en el Capítulo 10.

PROPINA

Como recordatorio, la contabilidad dentro de Lightning Network está en unidades de millisatoshis (milésimas de un satoshi), mientras que la contabilidad de Bitcoin está en satoshis. Cualquier monto en los HTLC son millisatoshis, que luego se redondean al satoshi más cercano en las transacciones de compromiso de Bitcoin.

HTLC en transacciones de compromiso

La información recibida es suficiente para que Bob cree un nuevo compromiso transacción. La nueva transacción de compromiso tiene las mismas dos salidas to_self y to_remote para el saldo de Alice y Bob, y una ***nueva*** salida representando el HTLC ofrecido por Alice.

Ya hemos visto la estructura básica de un HTLC en el Capítulo 8. El el script completo de un HTLC ofrecido se define en el TORNILLO #3: Transacciones, Salida HTLC ofrecida y se muestra en el Ejemplo 9-2.

Ejemplo 9-2. Script de salida HTLC ofrecido

```
# Revocación ❶
OP_DUP OP_HASH160 <RIPEMD160(SHA256(revocación de clave pública))> OP_EQUAL
OP_SI
    OP_CHECKSIG
OP_ELSE
<remote_HTLCpubkey> OP_SWAP OP_SIZE 32 OP_EQUAL
    OP_SI
        # Redención ❷
        OP_HASH160 <RIPEMD160(pago_hash)> OP_EQUALVERIFY
        2 OP_SWAP <local_HTLCpubkey> 2 OP_CHECKMULTISIG
    OP_ELSE
        # Reembolso ❸
        OP_DROP <cltv_expiry> OP_CHECKLOCKTIMEVERIFY OP_DROP
        OP_CHECKSIG
    OP_ENDIF
ON_ENDIF
```

❶ Alice puede canjear la primera cláusula del condicional OP_IF con un clave de revocación. Si este compromiso se revoca posteriormente, Alice tendrá un clave de revocación para reclamar esta salida en una transacción de penalización, tomando la equilibrio de todo el canal.

❷ La cláusula segunda es redimible por la preimagen (secreto de pago, o en nuestro ejemplo, el secreto de Dina) si es revelado. Esto le permite a Bob reclamar esta salida si tiene el secreto de Dina, lo que significa que ha logrado entregó el pago a Dina.

❸ La tercera y última cláusula es un reembolso del HTLC a Alice si el HTLC expira sin llegar a Dina. Está bloqueado en el tiempo con la expiración.

cltv_expiry. Esto asegura que el saldo de Alice no quede "atascado" en un HTLC que no se puede enrutar a Dina.

Hay tres formas de reclamar esta salida. Intente leer el script y vea si puede resolverlo (recuerde, es un lenguaje basado en pilas, por lo que las cosas aparecen "al revés").

Nuevo compromiso con la salida HTLC

Bob ahora tiene la información necesaria para agregar este script HTLC como salida adicional y crear una nueva transacción de compromiso. El nuevo compromiso de Bob tendrá 50.200 satoshis en la producción de HTLC. Esa cantidad provendrá del saldo del canal de Alice, por lo que el nuevo saldo de Alice será de 19 800 satoshis (70 000 – 50 200 = 19 800). Bob construye este compromiso como un “Compromiso #3” tentativo, que se muestra en la Figura 9-5.

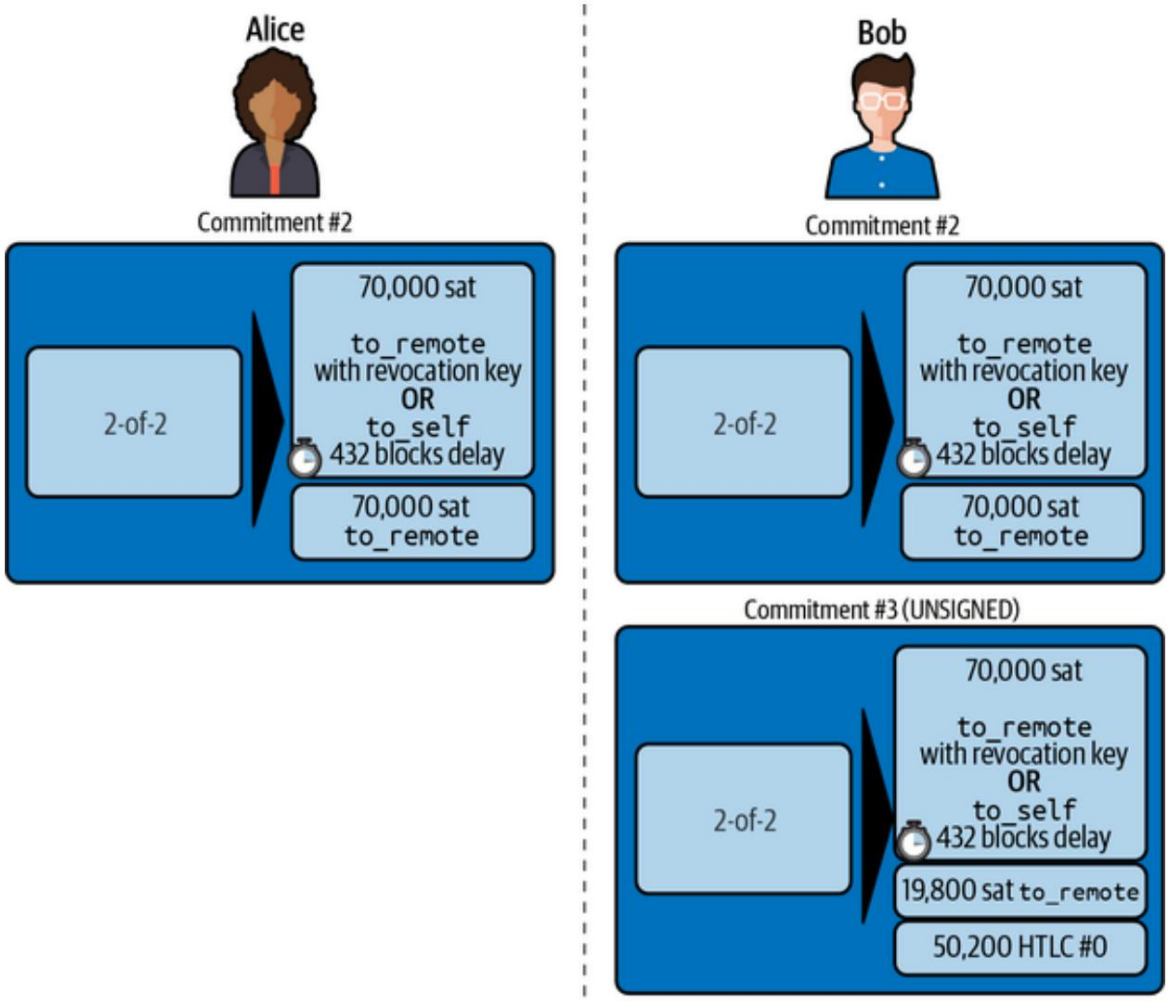

Figura 9-5. La nueva apuesta de Bob con una salida HTLC

Alicia se compromete

Poco después de enviar el mensaje update_add_htlc, se comprometerá con el nuevo estado del canal, de modo que Bob pueda agregar el HTLC de manera segura. Bob tiene la información de HTLC y ha construido un nuevo compromiso pero aún no tiene este nuevo compromiso firmado por Alice.

Alice envía el compromiso_firmado a Bob, con la firma del nuevo compromiso y del HTLC dentro. Vimos el mensaje de compromiso_firmado en el Capítulo 7, pero ahora podemos entender el resto de los campos. Como recordatorio, se muestra en el Ejemplo 9-3.

***Ejemplo 9-3. El mensaje** compromiso_firmado*

```
[id_canal:id_canal] [firma:firma]
[u16:num_htlcs]
[num_htlcs*firma:htlc_firma]
```

Los campos num_htlcs y htlc_signature ahora tienen más sentido:

num_htlcs

> Este es el número de HTLC que están pendientes en la transacción de compromiso. En nuestro ejemplo, solo un HTLC, el que ofrece Alice.

firma_htlc

> Esta es una matriz de firmas (num_htlcs de longitud), que contiene firmas para las salidas HTLC.

Alice puede enviar estas firmas sin dudarlo: siempre puede obtener un reembolso si el HTLC caduca sin ser enrutado a Dina.

Ahora, Bob tiene una nueva transacción de compromiso firmada, como se muestra en la Figura 9-6 .

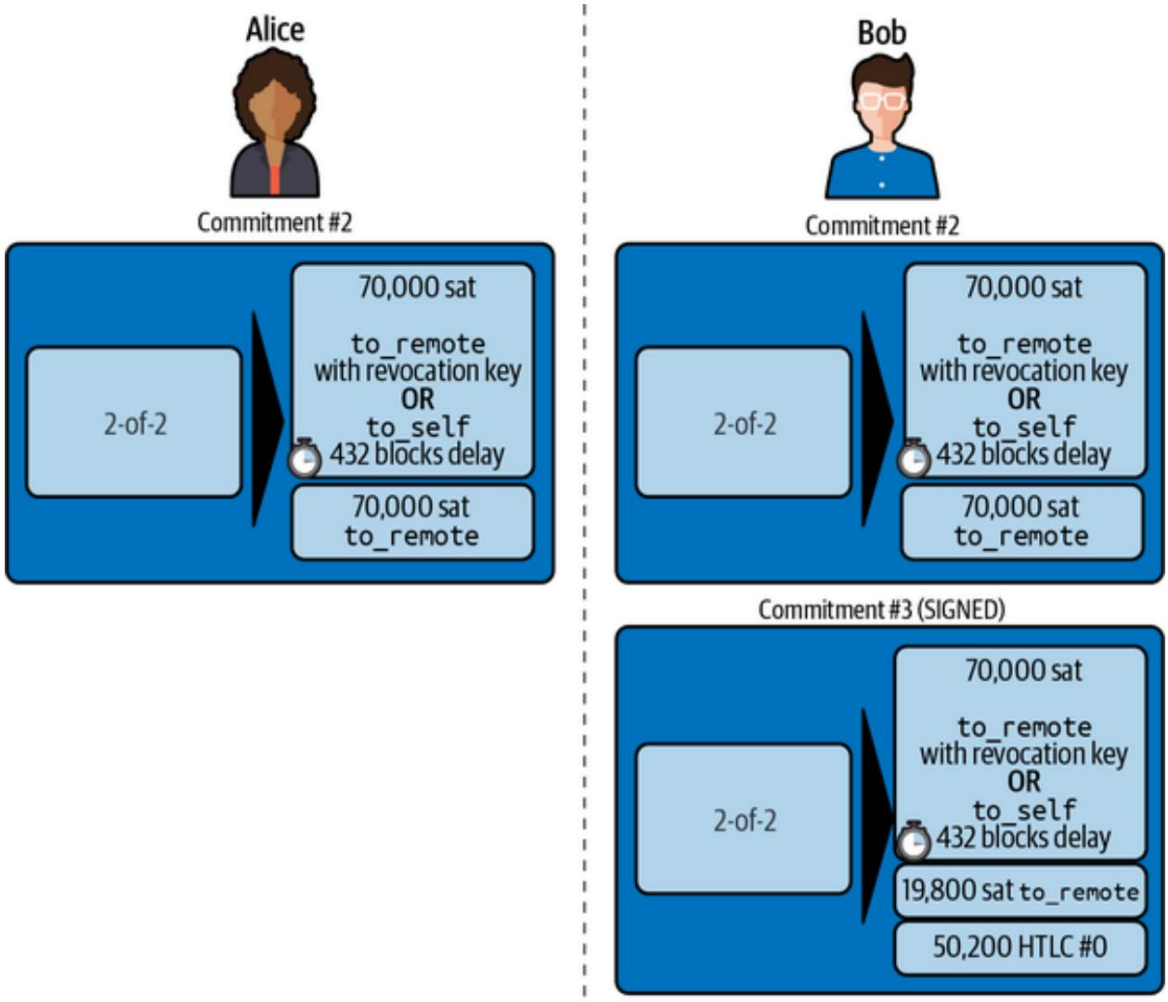

Figura 9-6. Bob tiene un nuevo compromiso firmado

Bob reconoce el nuevo compromiso y revoca el antiguo Una

Ahora que Bob tiene un nuevo compromiso firmado, necesita reconocerlo y revocar el antiguo compromiso. Lo hace enviando el mensaje revoke_and_ack, como vimos en el Capítulo 7 anteriormente. Como recordatorio, ese mensaje se muestra en el Ejemplo 9-4.

***Ejemplo 9-4. El mensaje** revocar y acusar recibo*

```
[channel_id:channel_id]
[32*byte:per_commitment_secret]
[point:next_per_commitment_point]
```

Bob envía el per_commitment_secret que le permite a Alice construir una clave de revocación para construir una transacción de penalización gastando el antiguo compromiso de Bob. Una vez que Bob haya enviado esto, nunca podrá publicar el "Compromiso n.º 2" sin correr el riesgo de recibir una multa por transacción y perder todo su dinero. Así, el antiguo compromiso queda efectivamente revocado.

Bob efectivamente movió el estado del canal hacia adelante, como se muestra en la Figura 9-7 .

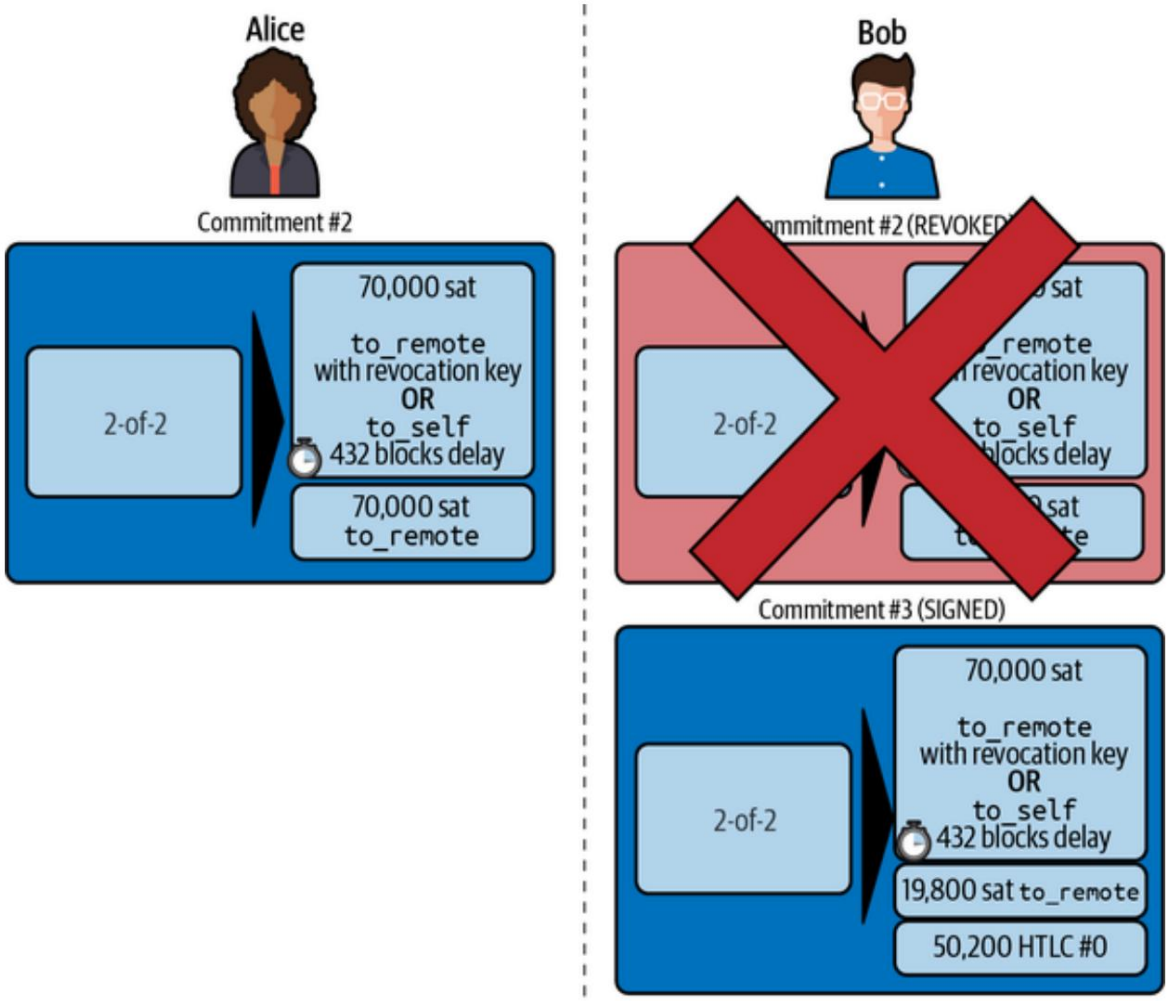

Figura 9-7. Bob ha revocado el antiguo compromiso.

A pesar de que Bob tiene una nueva transacción de compromiso (firmada) y una salida de HTLC interna, no puede considerar que su HTLC se haya configurado correctamente.

Primero necesita que Alice revoque su antiguo compromiso, porque de lo contrario, Alice puede revertir su saldo a 70,000 satoshis. Bob debe asegurarse de que Alice también tenga una transacción de compromiso que contenga el HTLC y haya revocado el compromiso anterior.

Por eso, si Bob no es el destinatario final de los fondos del HTLC, no debería reenviar el HTLC todavía ofreciendo un HTLC en el siguiente canal con Chan.

Alice ha construido una nueva transacción de compromiso de imagen especular que contiene el nuevo HTLC, pero aún debe ser firmada por Bob. Podemos verlo en la Figura 9-8.

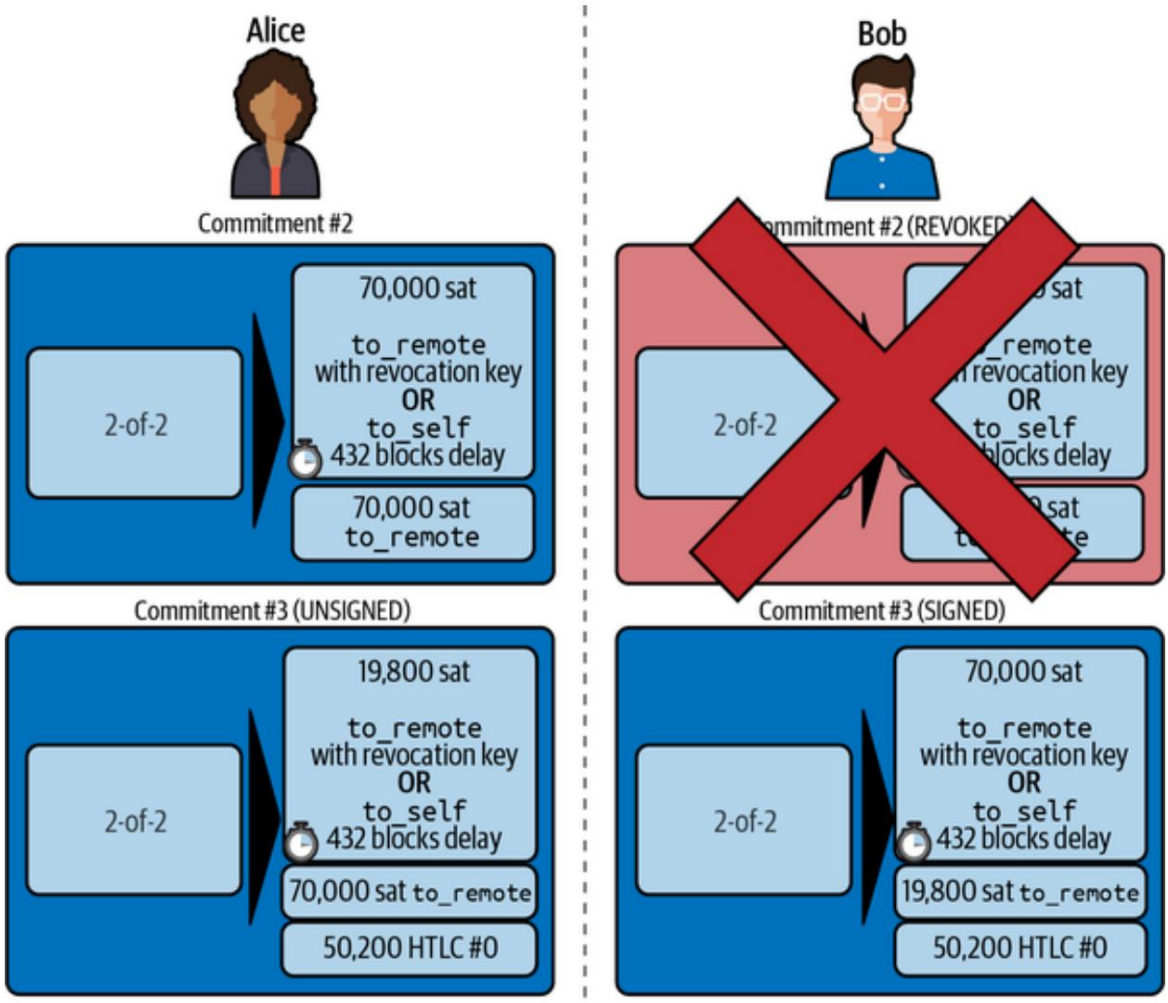

Figura 9-8. La nueva apuesta de Alice con una salida HTLC

Como describimos en el Capítulo 7, el compromiso de Alice es la imagen especular del de Bob, ya que contiene la construcción revocable, retrasada y asimétrica para la revocación y la aplicación de sanciones de compromisos antiguos. El saldo de 19 800 satoshi de Alice (después de deducir el valor HTLC) se retrasa y es revocable. El saldo de 70.000 satoshi de Bob se puede canjear inmediatamente.

A continuación, el flujo de mensajes para commit_signed y revocar_and_ack ahora se repite, pero en la dirección opuesta. Bob envía compromiso_firmado para firmar el nuevo compromiso de Alice, y Alice responde revocando su antiguo compromiso.

Para completar, revisemos rápidamente las transacciones de compromiso a medida que ocurre esta ronda de compromiso/revocación.

Bob se compromete

Bob ahora envía un compromiso_firmado a Alice, con sus firmas para la nueva transacción de compromiso de Alice, incluida la salida HTLC que ella ha agregado.

Ahora Alice tiene la firma para la nueva transacción de compromiso. El estado del canal se muestra en la Figura 9-9.

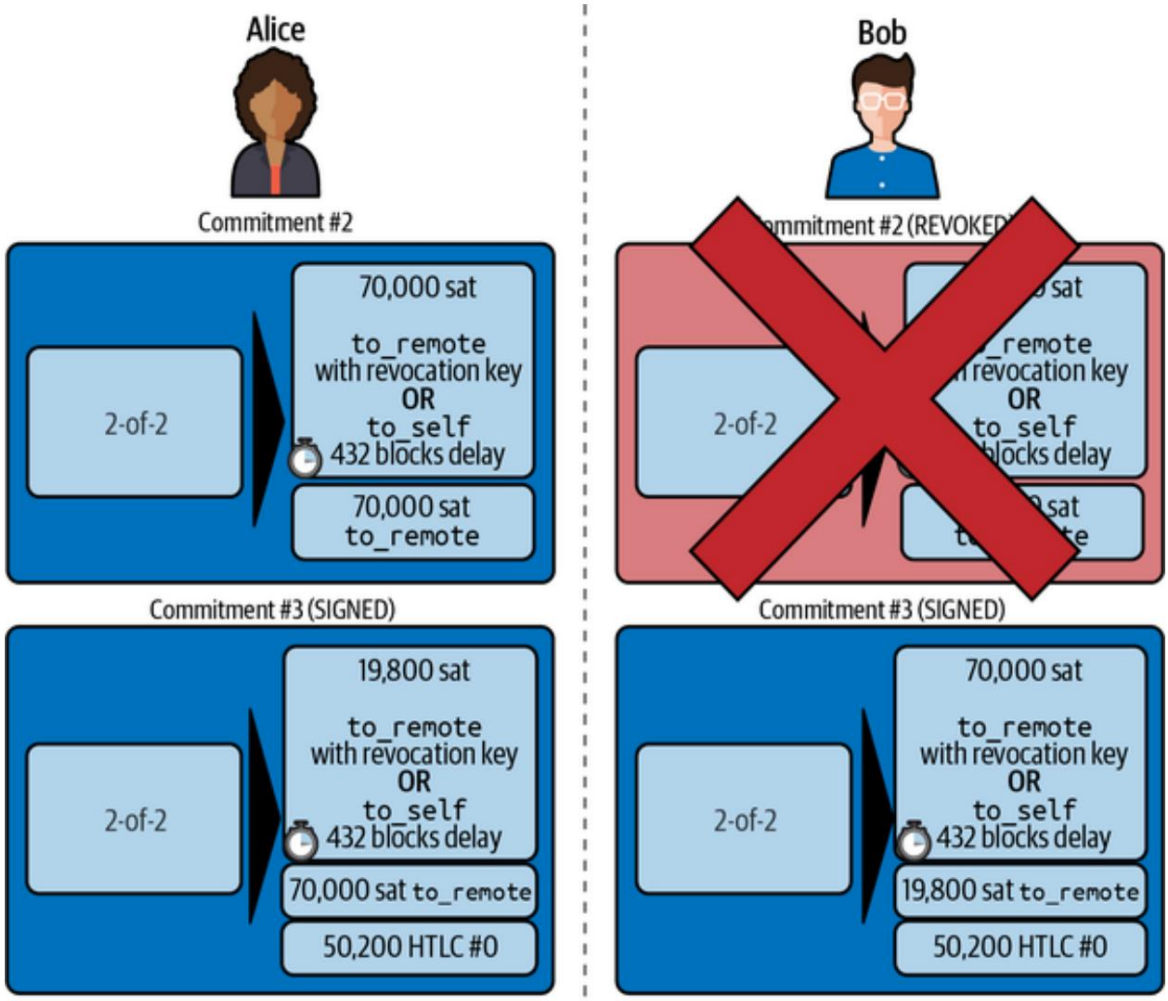

Figura 9-9. Alice tiene un nuevo compromiso firmado

Alice ahora puede reconocer el nuevo compromiso revocando el antiguo.
Alice envía el mensaje revoke_and_ack que contiene el per_commitment_point necesario que permitirá a Bob construir una clave de revocación y una transacción de penalización. Así, Alicia revoca su antiguo compromiso.

El estado del canal se muestra en la Figura 9-10.

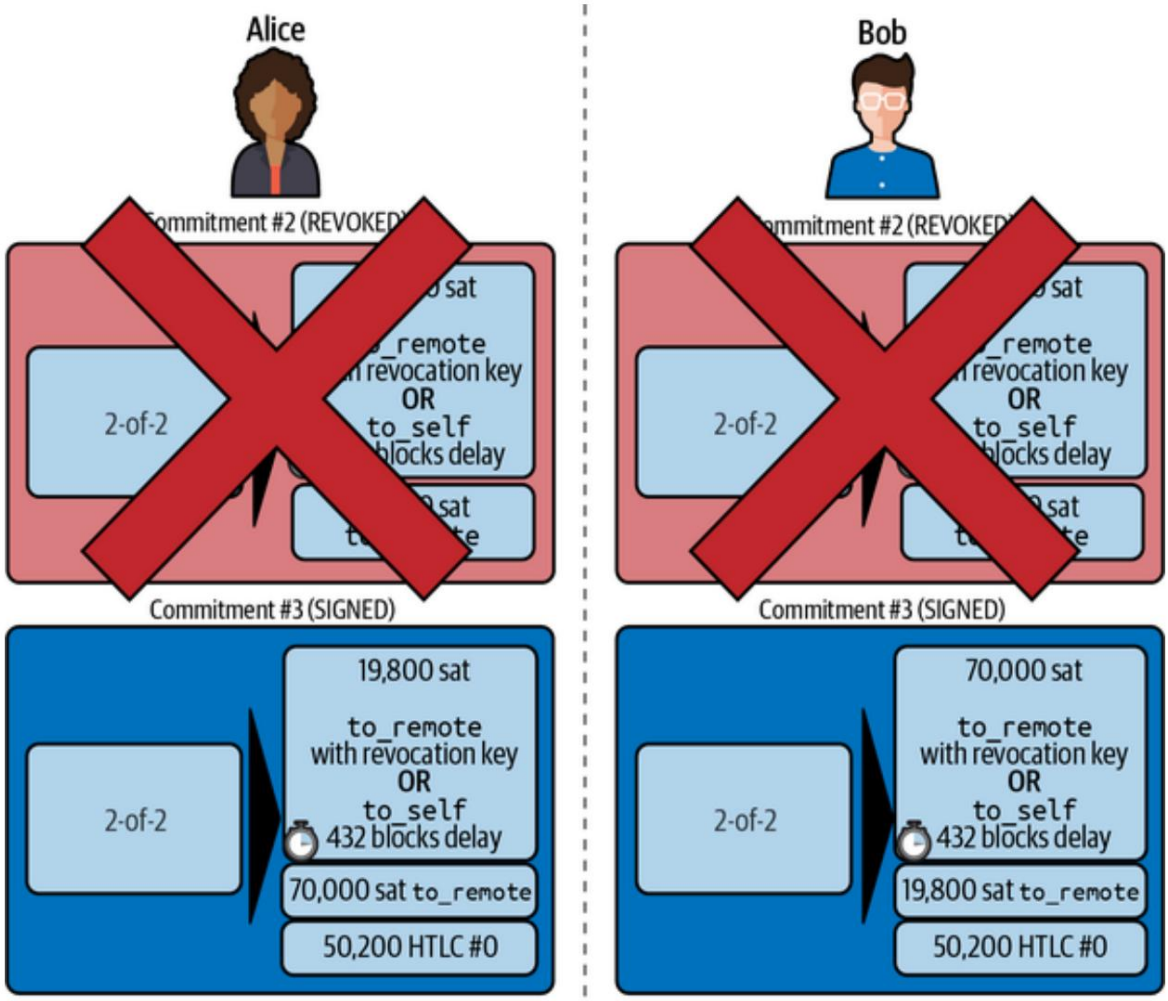

Figura 9-10. Alicia ha revocado el antiguo compromiso.

Múltiples HTLC

En cualquier momento, Alice y Bob pueden tener docenas o incluso cientos de HTLC en un solo canal. Cada HTLC se ofrece y se agrega a la transacción de compromiso como un resultado adicional. Por lo tanto, una transacción de comprometido siempre tiene dos salidas para los saldos del socio de canal y cualquier número de salidas HTLC, una por HTLC.

Como vimos en el mensaje commit_signed, hay una matriz de firmas HTLC para que se puedan transmitir varios compromisos HTLC al mismo tiempo.

El número máximo actual de HTLC permitidos en un canal es de 483 HTLC para dar cuenta del tamaño máximo de transacción de Bitcoin y garantizar que las transacciones de compromiso sigan siendo transacciones de Bitcoin válidas.

Como veremos en la siguiente sección, el máximo es solo para HTLC ***pendientes*** porque, una vez que se cumple un HTLC (o falla debido a un tiempo de espera/error), se elimina de la transacción de compromiso.

Cumplimiento de HTLC

Ahora Bob y Alice tienen una nueva transacción de compromiso con una salida HTLC adicional y hemos logrado un paso importante hacia la actualización de un canal de pago.

El nuevo balance de Alice y Bob aún no refleja que Alice haya enviado con éxito 50,200 satoshis a Bob.

Sin embargo, los HTLC ahora están configurados de manera que sea posible una liquidación segura a cambio del comprobante de pago.

Propagación HTLC

Supongamos que Bob continúa la cadena y establece un HTLC con Chan por 50 100 satoshis. El proceso será exactamente el mismo que acabamos de ver entre Alice y Bob. Bob enviará update_add_htlc a Chan, luego intercambiarán mensajes de compromiso_firmado y revocación y confirmación en dos rondas, haciendo avanzar su canal al siguiente estado.

A continuación, Chan hará lo mismo con Dina: ofrecerá un HTLC de 50 000 satoshi, comprometerá y revocará, etc. Sin embargo, Dina es la destinataria final del HTLC. Dina es la única que conoce el secreto de pago (la preimagen del hash de pago). ¡Por lo tanto, Dina puede cumplir el HTLC con Chan de inmediato!

Dina cumple el HTLC con Chan

Dina puede liquidar el HTLC enviando un mensaje update_ful fill_ htlc a Chan. El mensaje update_fulfill_htlc se define en BOLT #2: Peer Protocol, update_fulfill_htlc y se muestra aquí:

```
[channel_id:channel_id] [u64:id]
[32*byte:pago_preimagen]
```

Es un mensaje muy simple:

Canal ID

El ID del canal en el que se confirma el HTLC.

identificación

El ID del HTLC (comenzamos con 0 y aumentamos para cada HTLC en el canal).

pago_preimagen

El secreto que prueba que se realizó el pago y redime el HTLC. Este es el valor R que Dina compuso para producir el hash de pago en la factura a Alice.

Cuando Chan reciba este mensaje, comprobará inmediatamente si la preimagen de pago (llamémosla R) produce el hash de pago (llamémosla H) en el HTLC que le ofreció a Dina. Lo hash así:

H = RIPEMD160 (SHA-256 (R))

Si el resultado ***H*** coincide con el hash de pago en el HTLC, Chan puede hacer un pequeño baile de celebración. Este secreto largamente esperado se puede usar para canjear el HTLC y se transmitirá a lo largo de la cadena de canales de pago hasta Alice, resolviendo cada HTLC que formaba parte de este pago a Dina.

Volvamos al canal de Alice y Bob y obsérvalos relajar el HTLC. Para llegar allí, supongamos que Dina envió el update_fulfill_htlc

a Chan, Chan envió update_fulfill_htlc a Bob y Bob envió update_fulfill_htlc a Alice. La preimagen de pago se ha propagado hasta Alice.

Bob resuelve el HTLC con Alice

Cuando Bob envíe update_fulfill_htlc a Alice, contendrá la misma forma de pago_preimagen que Dina seleccionó para su factura. Ese pago_preimagen ha viajado hacia atrás a lo largo de la ruta de pago. En cada paso, channel_id será diferente e id (HTLC ID) puede ser diferente. ¡Pero la preimagen es la misma!

Alice también validará el pago_preimagen recibido de Bob. Comparará su hash con el hash de pago de HTLC en el HTLC que le ofreció a Bob. También encontrará que esta preimagen coincide con el hash en la factura de Dina. Esta es la prueba de que a Dina se le pagó.

El flujo de mensajes entre Alice y Bob se muestra en la figura 9-11.

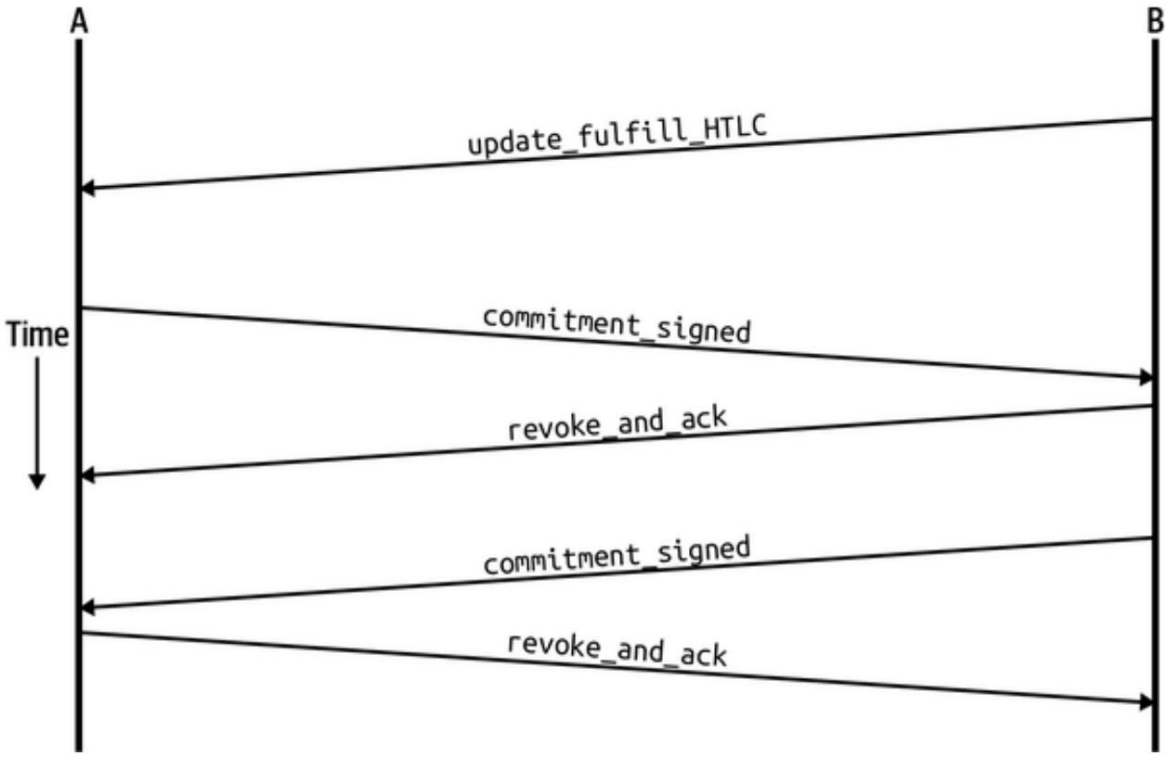

Figura 9-11. El flujo de mensajes de cumplimiento de HTLC

Tanto Alice como Bob ahora pueden eliminar el HTLC de las transacciones de compromiso y actualizar sus saldos de canal.

Crean nuevos compromisos (Compromiso #4), como se muestra en la Figura 9-12.

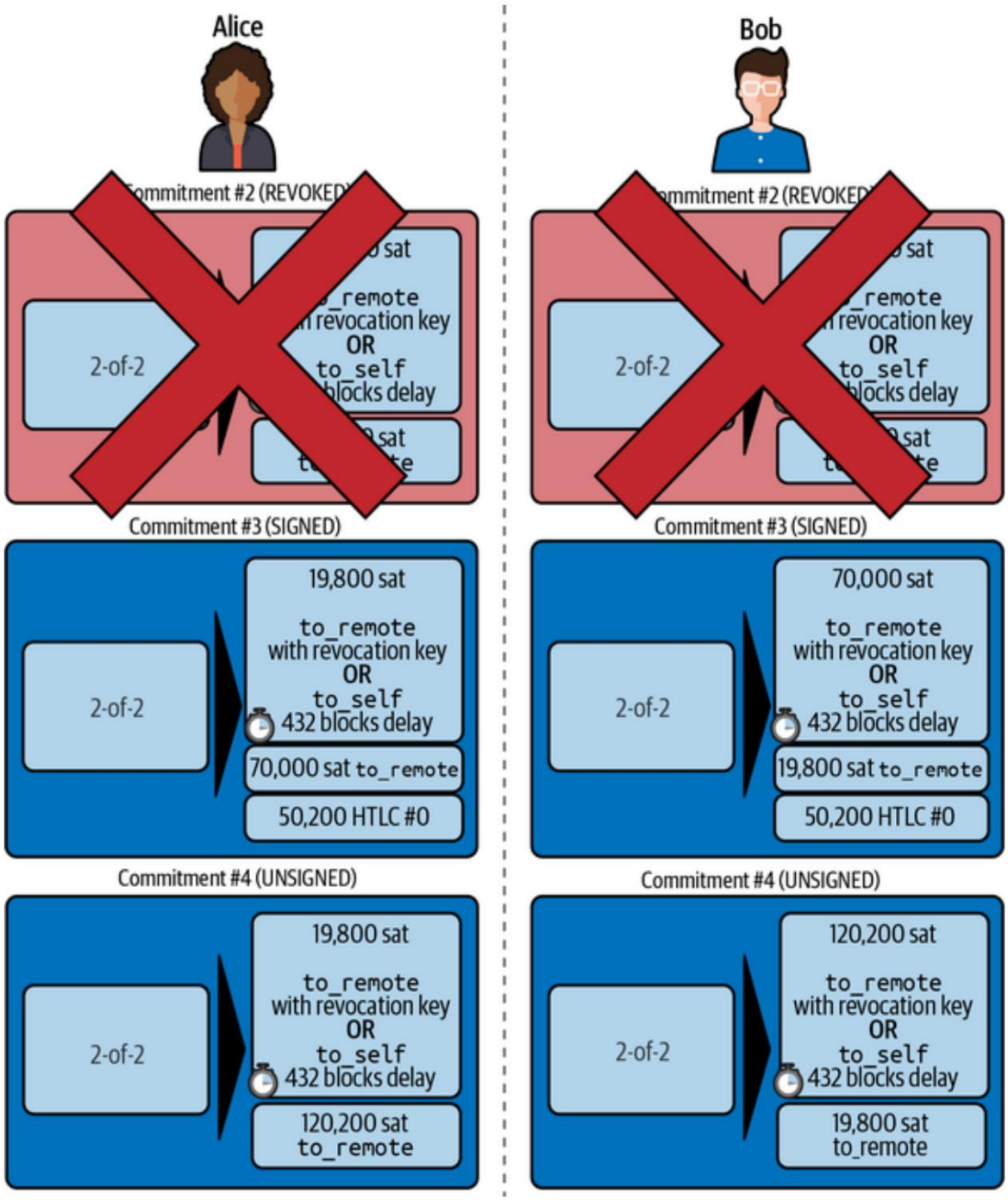

Figura 9-12. Se elimina el HTLC y se actualizan saldos en nuevos compromisos

Luego, completan dos rondas de compromiso y revocación. Primero, Alice envía commit_signed para firmar la nueva transacción de compromiso de Bob.
Bob responde con revoke_and_ack para revocar su antiguo compromiso.
Una vez que Bob ha movido el estado del canal hacia adelante, los compromisos se ven como los que vemos en la Figura 9-13.

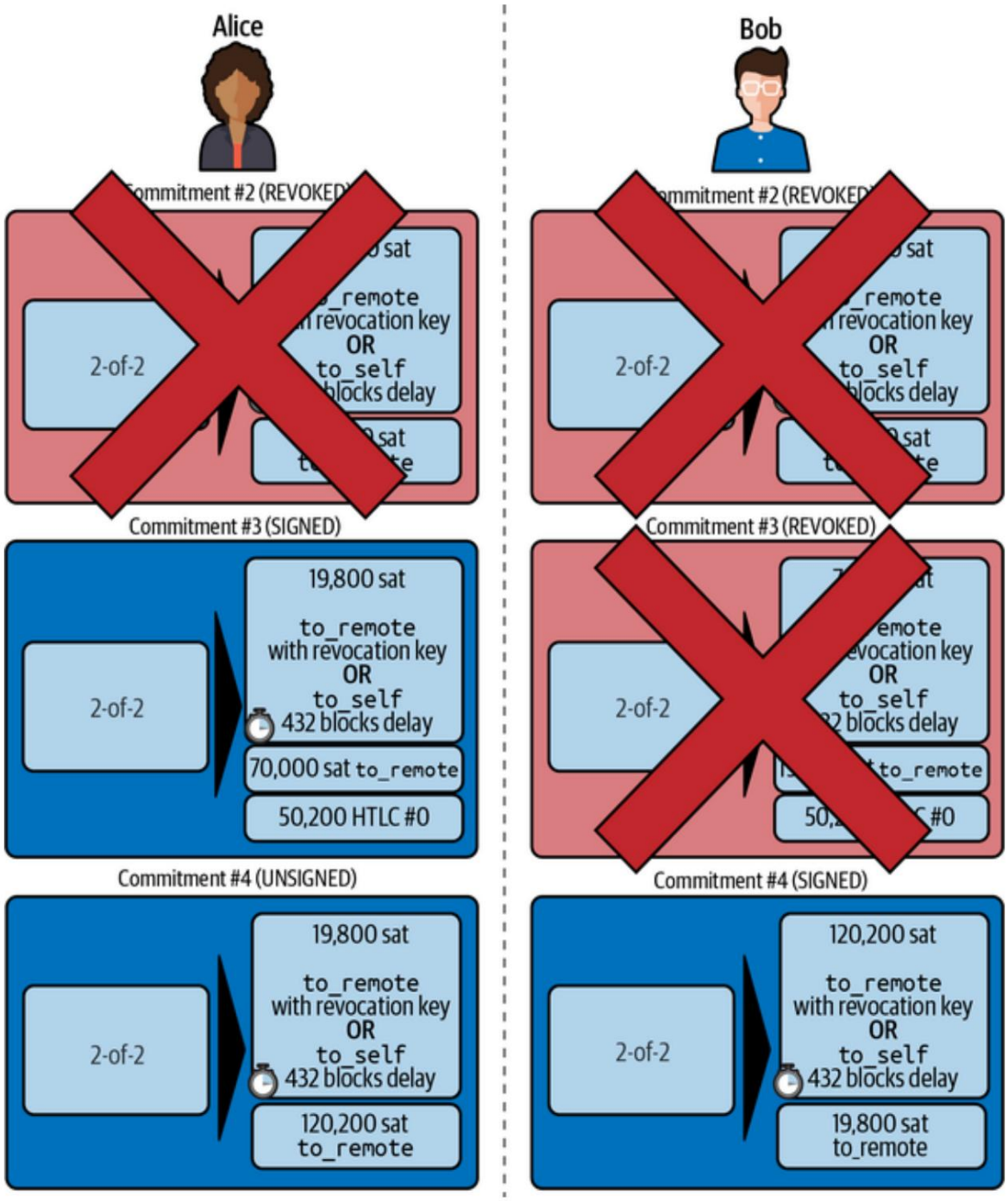

Figura 9-13. Alice firma el nuevo compromiso de Bob y Bob revocó el anterior.

Finalmente, Bob firma el compromiso de Alice enviándole a Alice un mensaje de compromiso_firmado. Entonces Alice reconoce y revoca su antiguo compromiso enviando revoke_and_ack a Bob. El resultado final es que tanto Alice como Bob han movido su estado de canal al Compromiso n.º 4, han eliminado el HTLC y han actualizado sus saldos.

Su estado de canal actual está representado por las transacciones de compromiso que se muestran en la Figura 9-14.

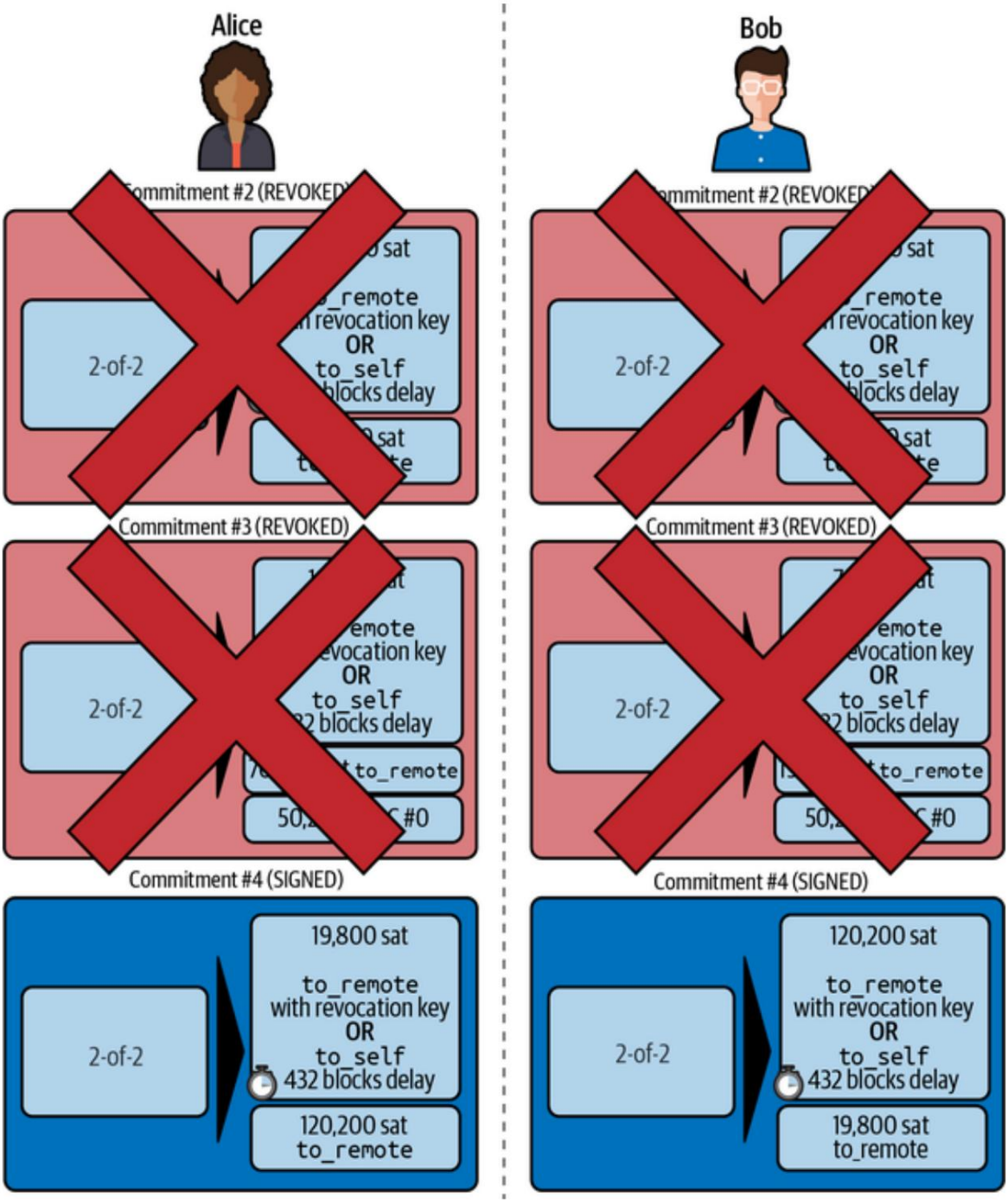

Figura 9-14. Alice y Bob liquidan el HTLC y actualizan los saldos

Eliminación de un HTLC debido a un error o vencimiento

Si un HTLC no se puede cumplir, se puede eliminar del compromiso del canal utilizando el mismo proceso de compromiso y revocación.

En lugar de update_fulfill_htlc, Bob enviaría un update_fail_htlc o update_fail_malformed_htlc. Estos dos mensajes se definen en BOLT #2: Peer Protocol, Eliminación de un HTLC.

El mensaje update_fail_htlc se muestra a continuación:

```
[id_canal:id_canal] [u64:id]
[u16:len] [len*byte:razón]
```

Es bastante autoexplicativo. El campo de motivo multibyte se define en el PERNO n.° 4: Enrutamiento cebolla, que describiremos en el capítulo 10.

Si Alice recibiera un update_fail_htlc de Bob, el proceso se desarrollaría de la misma manera: los dos socios de canal eliminarían el HTLC, crearían transacciones de compromiso actualizadas y pasarían por dos rondas de compromiso/revocación para mover el estado del canal al siguiente. compromiso. La única diferencia: los saldos finales volverían a ser lo que eran sin el HTLC, esencialmente reembolsando a Alice por el valor del HTLC.

Hacer un pago local

En este punto, comprenderá fácilmente por qué los HTLC se utilizan tanto para pagos remotos como locales. Cuando Alice le paga a Bob por un café, no solo actualiza el saldo del canal y se compromete con un nuevo estado. En cambio, el pago se realiza con un HTLC, de la misma manera que Alice le pagó a Dina. El hecho de que solo haya un salto de canal no hace ninguna diferencia. Funcionaría así:

1. Alice pide un café en la página de la tienda de Bob.
2. La tienda de Bob envía una factura con un hash de pago.
3. Alice construye un HTLC a partir de ese hash de pago.
4. Alice ofrece el HTLC a Bob con update_add_htlc.

5. Alice y Bob intercambian compromisos y revocaciones agregando el HTLC a sus transacciones de compromiso.

6. Bob envía update_fulfill_htlc a Alice con el pago preimagen

7. Alice y Bob intercambian compromisos y revocaciones eliminando el HTLC y la actualización de los balances de los canales.

Ya sea que un HTLC se reenvíe a través de muchos canales o simplemente se complete en un solo "salto" de canal, el proceso es exactamente el mismo

Conclusión

En este capítulo vimos cómo las transacciones de compromiso (del Capítulo 7) y los HTLC (del Capítulo 8) funcionan juntas. Vimos cómo se agrega un HTLC a una transacción de compromiso y cómo se cumple. Vimos cómo el sistema asimétrico, retrasado y revocable para hacer cumplir el estado del canal se extiende a los HTLC.

También vimos cómo un pago local y un pago enrutado de múltiples saltos se manejan de manera idéntica: usando HTLC.

En el próximo capítulo veremos el sistema de enrutamiento de mensajes encriptados llamado ***enrutamiento cebolla.***

Capítulo 10. Enrutamiento de cebolla

En este capítulo describiremos el mecanismo de enrutamiento de cebolla de Lightning Network. ¡ La invención del ***enrutamiento de cebolla*** precede a Lightning Network por 25 años! El enrutamiento cebolla fue inventado por investigadores de la Marina de los EE. UU. como un protocolo de seguridad de las comunicaciones. El enrutamiento de cebolla es el más famoso utilizado por Tor, la superposición de Internet enrutada de cebolla que permite a los investigadores, activistas, agentes de inteligencia y todos los demás usar Internet de forma privada y anónima.

En este capítulo, nos estamos enfocando en la parte de "Enrutamiento de cebolla basado en fuente (SPHINX)" de la arquitectura del protocolo Lightning, resaltada por un contorno en el centro (capa de enrutamiento) de la Figura 10-1.

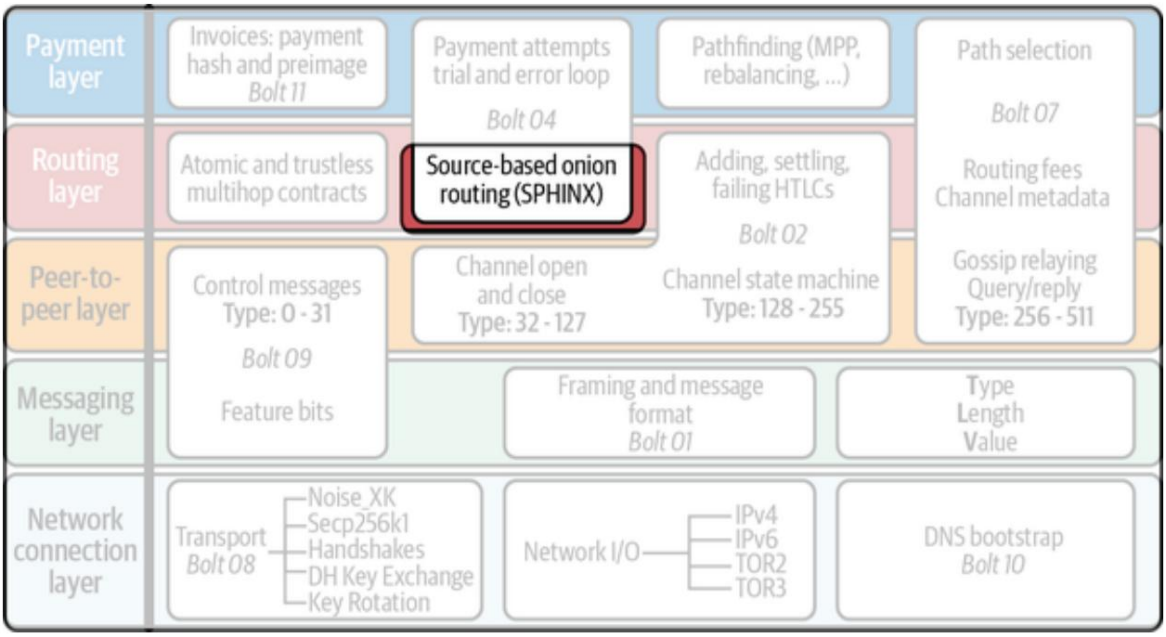

Figura 10-1. Enrutamiento de cebolla en el conjunto de protocolos Lightning

El enrutamiento cebolla describe un método de comunicación encriptada en el que el remitente del mensaje crea ***capas anidadas sucesivas de encriptación*** que cada nodo intermediario "despega" hasta que la capa más interna se entrega al destinatario previsto. El nombre "enrutamiento de cebolla" describe esto

uso de cifrado en capas que se quita una capa a la vez, como la piel de una cebolla.

Cada uno de los nodos intermediarios solo puede "pelar" una capa y ver quién es el siguiente en la ruta de comunicación. El enrutamiento cebolla garantiza que nadie, excepto el remitente, conozca el destino o la longitud de la ruta de comunicación. Cada intermediario solo conoce el salto anterior y el siguiente.

Lightning Network utiliza una implementación del protocolo de enrutamiento de cebolla basado en ***Sphinx,*** [1] desarrollado en 2009 por George Danezis e Ian Goldberg.

La implementación del enrutamiento cebolla en Lightning Network se define en BOLT #4: Protocolo de enrutamiento cebolla.

Un ejemplo físico que ilustra la cebolla

Enrutamiento

Hay muchas formas de describir el enrutamiento de cebolla, pero una de las más fáciles es usar el equivalente físico de los sobres sellados. Un sobre representa una capa de encriptación, que solo permite que el destinatario designado lo abra y lea el contenido.

Digamos que Alice quiere enviar una carta secreta a Dina, indirectamente a través de algunos intermediarios.

Seleccionar una ruta

Lightning Network utiliza ***el enrutamiento de origen,*** lo que significa que la ruta de pago es seleccionada y especificada por el remitente, y solo por el remitente. En este ejemplo, la carta secreta de Alice a Dina será el equivalente a un pago.
Para asegurarse de que la carta llegue a Dina, Alice creará un camino de ella a Dina, utilizando a Bob y Chan como intermediarios.

PROPINA

Puede haber muchos caminos que hagan posible que Alice alcance a Dina. Explicaremos el proceso de selección de la ruta ***óptima*** en el Capítulo 12. Por ahora, supondremos que la ruta seleccionada por Alice usa a Bob y Chan como intermediarios para llegar a Dina.

Como recordatorio, la ruta seleccionada por Alice se muestra en la Figura 10-2.

Figura 10-2. Camino: Alice a Bob a Chan a Dina

Veamos cómo Alice puede usar este camino sin revelar información a los intermediarios Bob y Chan.

ENRUTAMIENTO BASADO EN FUENTE

El enrutamiento basado en la fuente no es la forma en que los paquetes se enrutan normalmente en Internet hoy en día, aunque el enrutamiento en la fuente era posible en los primeros días. El enrutamiento de Internet se basa en ***la conmutación*** de paquetes en cada nodo de enrutamiento intermediario. Un paquete IPv4, por ejemplo, incluye las direcciones IP del remitente y del destinatario, y todos los demás nodos de enrutamiento IP deciden cómo reenviar cada paquete hacia el destino. Sin embargo, la falta de privacidad en un mecanismo de enrutamiento de este tipo, donde cada nodo intermediario ve al remitente y al destinatario, hace que sea una mala elección para usar en una red de pago.

Construyendo las capas

Alice comienza escribiendo una carta secreta a Dina. Luego sella la carta dentro de un sobre y escribe “Para Dina” en el exterior (vea la Figura 10-3). El sobre representa el cifrado con la clave pública de Dina para que solo Dina pueda abrir el sobre y leer la carta.

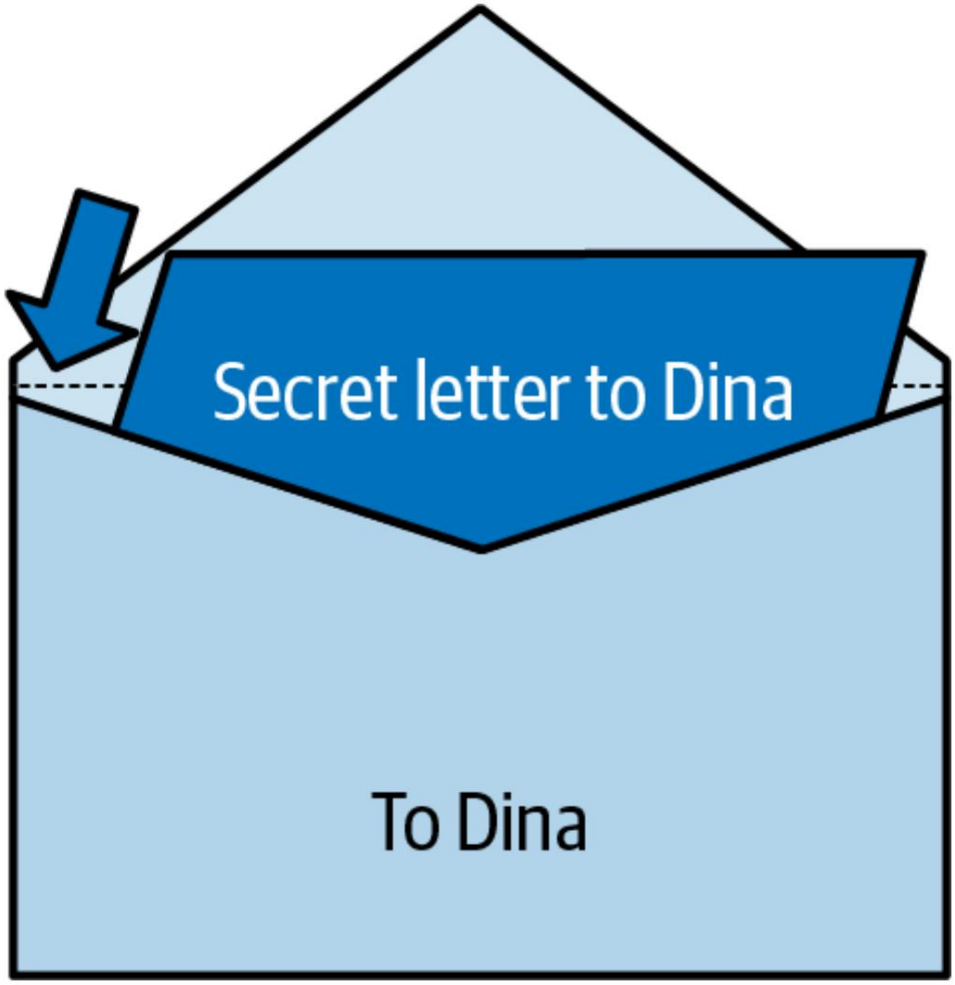

Figura 10-3. La carta secreta de Dina, sellada en un sobre.

La carta de Dina será entregada a Dina por Chan, quien está inmediatamente antes de Dina en el "camino". Entonces, Alice pone el sobre de Dina dentro de un sobre dirigido a Chan (vea la Figura 10-4). La única parte que Chan puede leer es el destino (instrucciones de ruta): "Para Dina". Sellar esto dentro de un sobre dirigido a Chan representa cifrarlo con la clave pública de Chan para que solo Chan pueda leer la dirección del sobre. Chan todavía no puede abrir el sobre de Dina. Todo lo que ve son las instrucciones en el exterior (la dirección).

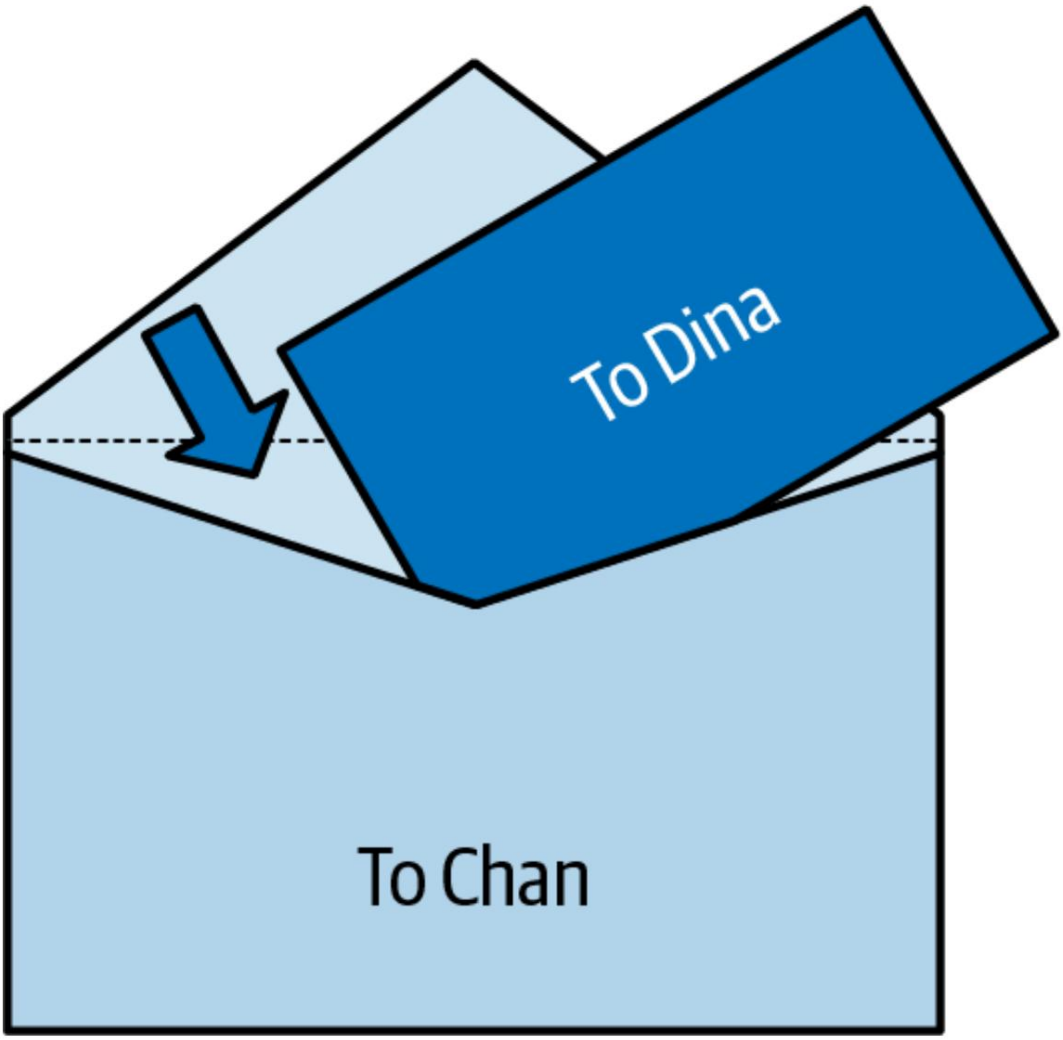

Figura 10-4. Sobre de Chan, que contiene el sobre sellado de Dina.

Ahora, Bob le entregará esta carta a Chan. Entonces Alice lo pone dentro de un sobre dirigido a Bob (vea la figura 10-5). Como antes, el sobre representa un mensaje encriptado para Bob que solo Bob puede leer. Bob solo puede leer el exterior del sobre de Chan (la dirección), por lo que sabe que debe enviárselo a Chan.

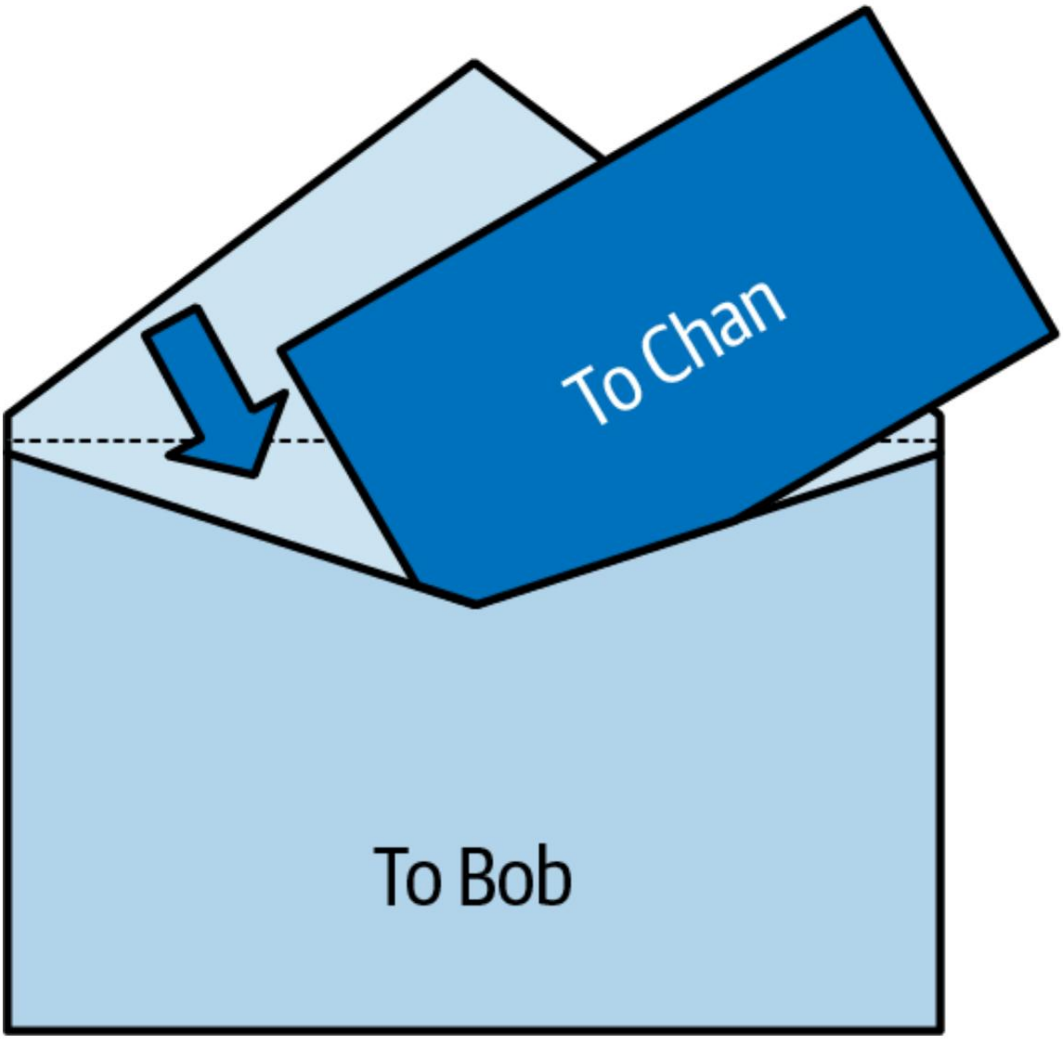

Figura 10-5. Sobre de Bob, que contiene el sobre sellado de Chan.

Ahora, si pudiéramos mirar a través de los sobres (¡con rayos X!) veríamos los sobres anidados uno dentro del otro, como se muestra en la Figura 10-6.

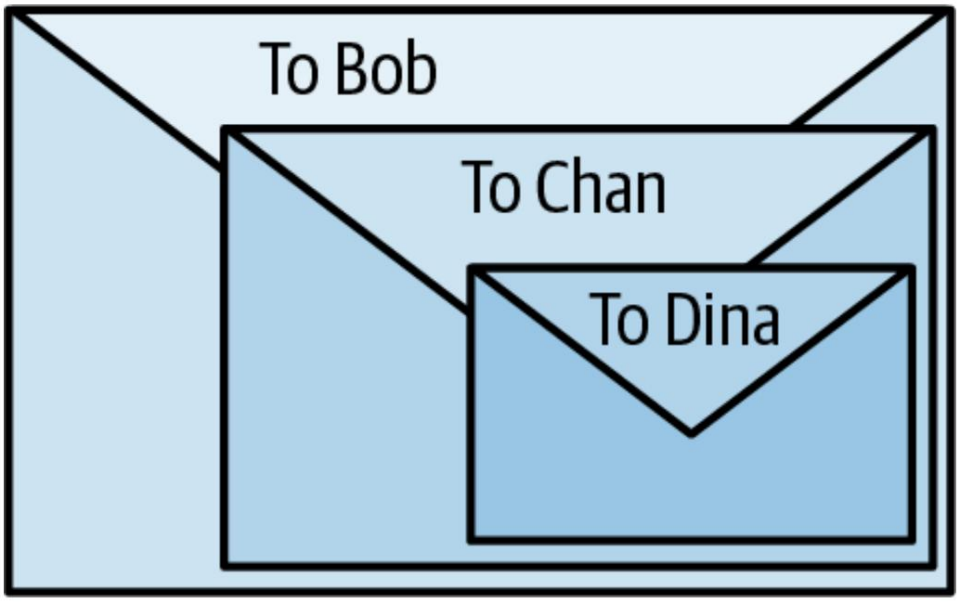

Figura 10-6. Sobres anidados

Pelar las capas

Alice ahora tiene un sobre que dice "Para Bob" en el exterior. Representa un mensaje cifrado que solo Bob puede abrir (descifrar). Alice ahora comenzará el proceso enviándole esto a Bob. El proceso completo se muestra en la Figura 10-7.

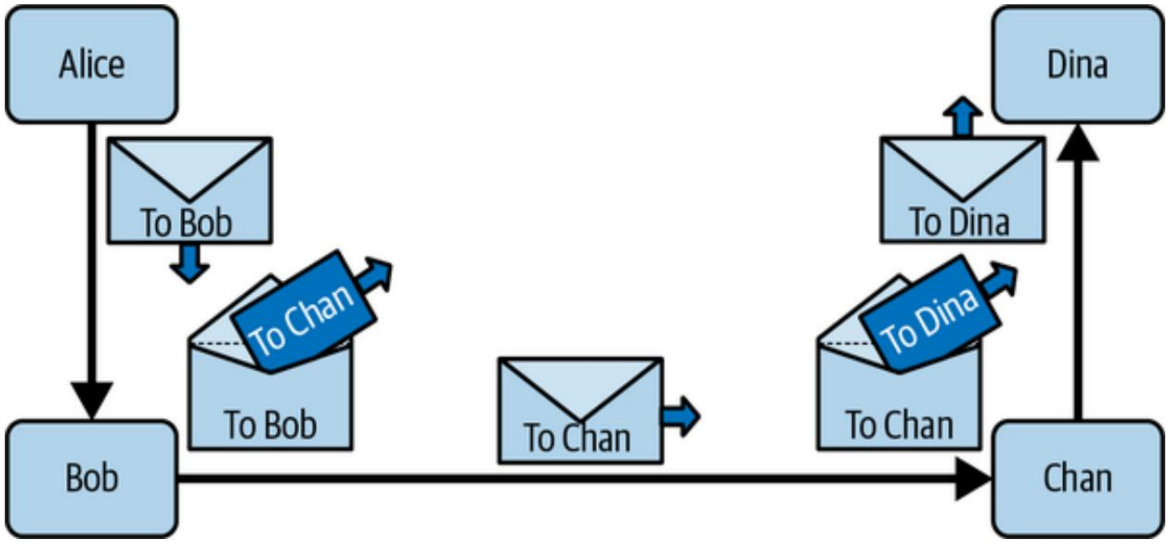

Figura 10-7. enviar los sobres

Como puede ver, Bob recibe el sobre de Alice. Sabe que vino de Alice, pero no sabe si Alice es el remitente original o simplemente alguien

reenvío de sobres. Lo abre para encontrar un sobre dentro que dice "Para Chan". Como está dirigido a Chan, Bob no puede abrirlo. No sabe qué hay dentro y no sabe si Chan está recibiendo una carta u otro sobre para reenviar. Bob no sabe si Chan es el destinatario final o no. Bob le pasa el sobre a Chan.

Chan recibe el sobre de Bob. Él no sabe que vino de Alice. No sabe si Bob es un intermediario o el remitente de una carta.
Chan abre el sobre y encuentra otro sobre dentro con la dirección "Para Dina", que no puede abrir. Chan se lo reenvía a Dina, sin saber si Dina es el destinatario final.

Dina recibe un sobre de Chan. Al abrirlo, encuentra una carta adentro, por lo que ahora sabe que ella es la destinataria de este mensaje. ¡Ella lee la carta, sabiendo que ninguno de los intermediarios sabe de dónde vino y nadie más ha leído su carta secreta!

Esta es la esencia del enrutamiento de cebolla. El remitente envuelve un mensaje en capas, especificando exactamente cómo se enrutará y evitando que cualquiera de los intermediarios obtenga información sobre la ruta o la carga útil.
Cada intermediario pela una capa, solo ve una dirección de reenvío y no sabe nada más que el salto anterior y el siguiente en la ruta.

Ahora, veamos los detalles de la implementación del enrutamiento cebolla en Lightning Network.

Introducción al enrutamiento cebolla de HTLC

El enrutamiento de cebolla en Lightning Network parece complejo a primera vista, pero una vez que comprende el concepto básico, es realmente bastante simple.

Desde una perspectiva práctica, Alice le dice a cada nodo intermediario qué HTLC debe configurar con el siguiente nodo en la ruta.

El primer nodo, que es el remitente del pago o Alice en nuestro ejemplo, se denomina ***nodo de origen.*** El último nodo, que es el destinatario del pago o Dina en nuestro ejemplo, se denomina ***nodo final.***

Cada nodo intermediario, o Bob y Chan en nuestro ejemplo, se denomina ***salto.*** Cada salto debe configurar un ***HTLC saliente*** para el siguiente salto. La información comunicada a cada salto por Alice se denomina ***carga útil*** de salto o ***datos de salto.*** El mensaje que se enruta de Alice a Dina se denomina ***cebolla*** y consta de mensajes ***de carga útil de salto*** cifrados o ***de datos*** de salto cifrados para cada salto.

Ahora que conocemos la terminología utilizada en el enrutamiento de cebolla Lightning, repitamos la tarea de Alice: Alice debe construir una cebolla con datos de salto, diciéndole a cada salto cómo construir un HTLC saliente para enviar un pago al nodo final (Dina).

Alice selecciona el camino

Del Capítulo 8 sabemos que Alice enviará un pago de 50,000 satoshi a Dina a través de Bob y Chan. Este pago se transmite a través de una serie de HTLC, como se muestra en la Figura 10-8.

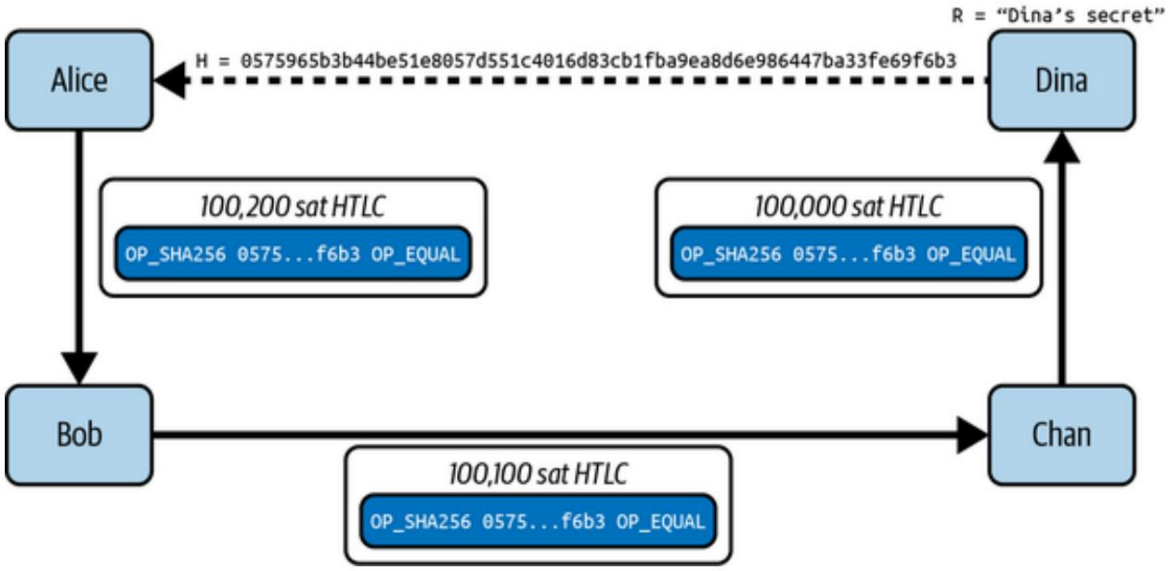

Figura 10-8. Ruta de pago con HTLC de Alice a Dina

Como veremos en el Capítulo 11, Alice puede construir este camino hacia Dina porque los nodos Lightning anuncian sus canales a toda la Red Lightning utilizando el Protocolo Lightning Gossip. Después del anuncio inicial del canal, Bob y Chan enviaron cada uno un mensaje adicional

mensaje channel_update con su tarifa de enrutamiento y expectativas de bloqueo de tiempo para el enrutamiento de pago.

De los anuncios y actualizaciones, Alice conoce la siguiente información sobre los canales entre Bob, Chan y Dina:

- Un short_channel_id (ID de canal corto) para cada canal, que Alice puede usar para hacer referencia al canal al construir la ruta

- Un cltv_expiry_delta (delta de bloqueo de tiempo), que Alice puede agregar al tiempo de vencimiento para cada HTLC

- Fee_base_msat y fee_proportional_millionths, que Alice puede usar para calcular la tarifa de enrutamiento total esperada por ese nodo para la retransmisión en ese canal.

En la práctica, también se intercambia otra información, como los HTLC más grandes (htlc_maximum_msat) y más pequeños (htlc_minimum_msat) que transportará un canal, pero estos no se usan tan directamente durante la construcción de la ruta de cebolla como los campos anteriores.

Alice utiliza esta información para identificar los nodos, los canales, las tarifas y los bloqueos de tiempo para la siguiente ruta detallada, que se muestra en la figura 10-9.

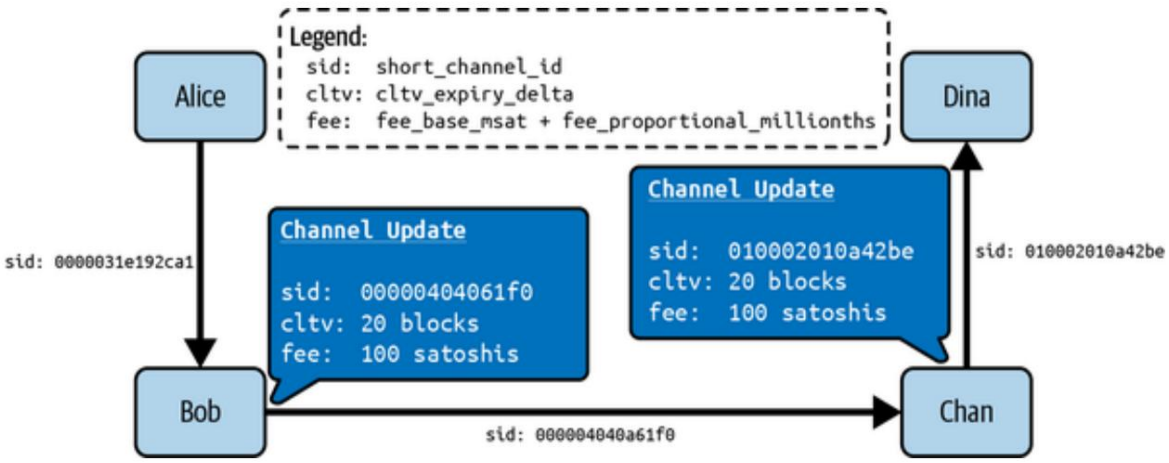

Figura 10-9. Una ruta detallada construida a partir de información de canal y nodo comentada

Alice ya conoce su propio canal con Bob y, por lo tanto, no necesita esta información para construir la ruta. Tenga en cuenta también que Alice no necesitaba una actualización de canal de Dina porque tiene la actualización de Chan para ese último canal en la ruta.

Alice construye las cargas útiles

Hay dos formatos posibles que Alice puede usar para la información comunicada a cada salto: un formato heredado de longitud fija llamado ***datos de salto*** y un formato basado en Tipo-Longitud-Valor (TLV) más flexible llamado ***carga útil de salto.*** El formato de mensaje TLV se explica con más detalle en "Tipo Longitud-Valor Formato". Ofrece flexibilidad al permitir que se agreguen campos al protocolo a voluntad.

> **NOTA**
>
> Ambos formatos se especifican en BOLT #4: Protocolo de enrutamiento cebolla, estructura de paquetes.

Alice comenzará a generar los datos de salto desde el final de la ruta hacia atrás: Dina, Chan y luego Bob.

Carga útil del nodo final para Dina

Alice primero construye la carga útil que se entregará a Dina. Dina no construirá un "HTLC saliente", porque Dina es el nodo final y el destinatario del pago. Por esta razón, la carga útil de Dina es diferente a la de todos los demás (usa solo ceros para short_channel_id), pero solo Dina lo sabrá porque estará encriptado en la capa más interna de la cebolla. Esencialmente, esta es la "carta secreta a Dina" que vimos en nuestro ejemplo de sobre físico.

La carga útil de salto para Dina debe coincidir con la información de la factura generada por Dina para Alice y contendrá (al menos) los siguientes campos en formato TLV:

amt_to_forward

El monto de este pago en millisatoshis. Si esta es solo una parte de un pago de varias partes, el monto es menor que el total. De lo contrario, este es un pago único y completo y es igual al monto de la factura y al valor total_msat.

valor_cltv_saliente

El límite de tiempo de vencimiento del pago establecido en el valor min_final_cltv_expiry en la factura.

pago_secreto

Un valor secreto especial de 256 bits de la factura, que permite a Dina reconocer este pago entrante. Esto también evita una clase de sondeo que anteriormente hacía inseguras las facturas de valor cero. El sondeo por parte de los nodos intermedios se mitiga ya que este valor se cifra ***solo*** para el destinatario, lo que significa que no pueden reconstruir un paquete final que "parece" legítimo.

total_msat

El importe total que coincide con la factura. Esto se puede omitir si solo hay una parte, en cuyo caso se supone que coincide con amt_to_forward y debe ser igual al monto de la factura.

La factura que Alice recibió de Dina especificaba la cantidad como 50 000 satoshis, que son 50 000 000 millisatoshis. Dina especificó la caducidad mínima para el pago min_final_cltv_expiry como 18 bloques (3 horas, con bloques de Bitcoin promedio de 10 minutos). En el momento en que Alice intenta realizar el pago, digamos que la cadena de bloques de Bitcoin ha registrado 700 000 bloques. Entonces, Alice debe establecer outgoing_cltv_value en una altura de bloque ***mínima*** de 700,018.

Alice construye la carga útil de salto para Dina de la siguiente manera:

amt_to_forward: 50,000,000
outgoing_cltv_value: 700,018

pago_secreto:
fb53d94b7b65580f75b98f10...03521bdab6d519143cd521d1b3826
total_msat: 50,000,000

Alice lo serializa en formato TLV, como se muestra (simplificado) en la figura 10-10.

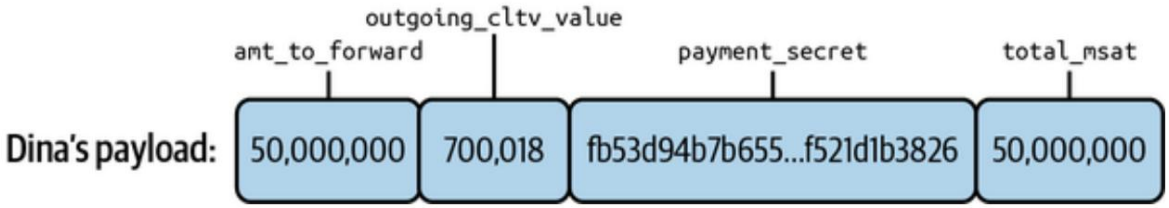

Figura 10-10. La carga útil de Dina está construida por Alice.

Carga útil de lúpulo para Chan

A continuación, Alice construye la carga útil de salto para Chan. Esto le dirá a Chan cómo configurar un HTLC saliente para Dina.

La carga útil de salto para Chan incluye tres campos: short_channel_id, amt_to_forward y outgoing_cltv_value:

short_channel_id: 010002010a42be
amt_to_forward: 50,000,000 outgoing_cltv_value:
700,018

Alice serializa esta carga útil en formato TLV, como se muestra (simplificado) en la figura 10-11.

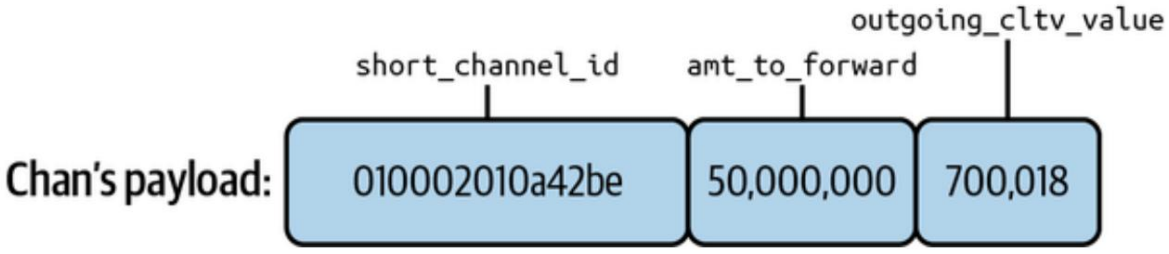

Figura 10-11. La carga útil de Chan está construida por Alice.

Carga útil de salto para Bob

Finalmente, Alice construye la carga útil de salto para Bob, que también contiene los mismos tres campos que la carga útil de salto para Chan, pero con valores diferentes:

short_channel_id: 000004040a61f0 amt_to_forward:
50,100,000 outgoing_cltv_value: 700,038

Como puede ver, el campo amt_to_forward es 50 100 000 milisatoshis o 50 100 satoshis. Eso es porque Chan espera una tarifa de 100 satoshis para enrutar un pago a Dina. Para que Chan "gane" esa tarifa de enrutamiento, el HTLC entrante de Chan debe ser 100 satoshis más que el HTLC saliente de Chan. Dado que el HTLC entrante de Chan es el HTLC saliente de Bob, las instrucciones para Bob reflejan la tarifa que gana Chan. En términos simples, se le debe decir a Bob que envíe 50 100 satoshi a Chan, para que Chan pueda enviar 50 000 satoshi y quedarse con 100 satoshi.

De manera similar, Chan espera un delta de bloqueo de tiempo de 20 bloques. Entonces, el HTLC entrante de Chan debe expirar 20 bloques ***más tarde*** que el HTLC saliente de Chan. Para lograr esto, Alice le dice a Bob que haga que su HTLC saliente a Chan expire a la altura del bloque 700,038-20 bloques más tarde que el HTLC de Chan a Dina.

PROPINA

Las tarifas y las expectativas delta de bloqueo de tiempo para un canal se establecen por la diferencia entre los HTLC entrantes y salientes. Dado que el HTLC entrante lo crea el ***nodo anterior,*** la tarifa y el delta de bloqueo de tiempo se establecen en la carga útil de cebolla para ese nodo anterior. A Bob se le dice cómo hacer un HTLC que cumpla con las expectativas de Chan y timelock.

Alice serializa esta carga útil en formato TLV, como se muestra (simplificado) en la figura 10-12.

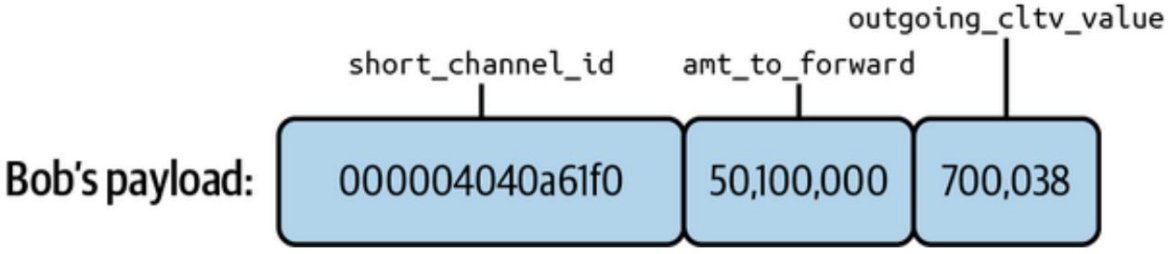

Figura 10-12. La carga útil de Bob está construida por Alice.

Cargas útiles de salto finalizadas

Alice ahora ha construido las cargas útiles de tres saltos que se envolverán en una cebolla.
En la Figura 10-13 se muestra una vista simplificada de las cargas útiles .

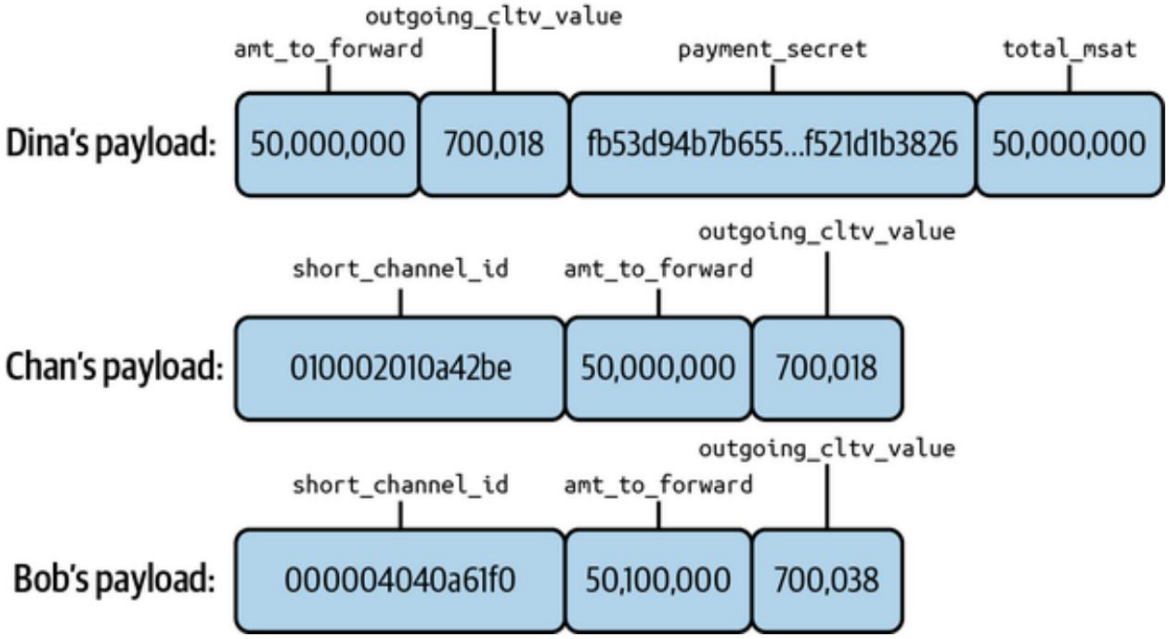

Figura 10-13. Cargas útiles de salto para todos los saltos

Generación de claves

Alice ahora debe generar varias claves que se utilizarán para cifrar las distintas capas de la cebolla.

Con estas claves, Alice puede lograr un alto grado de privacidad e integridad:

- Alice puede cifrar cada capa de la cebolla para que solo el destinatario previsto pueda leerla.
- Todo intermediario puede comprobar que el mensaje no se modifica.
- Nadie en el camino sabrá quién envió esta cebolla o hacia dónde va. Alice no revela su identidad como remitente ni la identidad de Dina como receptora del pago.

- Cada salto solo aprende sobre el salto anterior y el siguiente.
- Nadie puede saber qué tan largo es el camino, o en qué parte del camino se encuentran. son.

> **ADVERTENCIA**
>
> Como una cebolla picada, los siguientes detalles técnicos pueden hacerte llorar. Siéntase libre de pasar a la siguiente sección si se confunde. Vuelva a esto y lea TORNILLO #4: Enrutamiento de cebolla, construcción de paquetes, si desea obtener más información.

La base de todas las claves utilizadas en la cebolla es un ***secreto compartido*** que Alice y Bob pueden generar de forma independiente utilizando el algoritmo Elliptic Curve Diffie-Hellman (ECDH). A partir del secreto compartido (ss), pueden generar de forma independiente cuatro claves adicionales denominadas rho, mu, um y pad:

ro

Se utiliza para generar un flujo de bytes aleatorios a partir de un cifrado de flujo (utilizado como CSPRNG). Estos bytes se utilizan para cifrar/descifrar el cuerpo del mensaje, así como los bytes cero de relleno durante el procesamiento del paquete Sphinx.

en

Se utiliza en el código de autenticación de mensajes basado en hash (HMAC) para la verificación de integridad/autenticidad.

a

Se utiliza en el informe de errores.

almohadilla

Se utiliza para generar bytes de relleno para rellenar la cebolla a una longitud fija.

La relación entre las diversas claves y cómo se generan se diagrama en la figura 10-14.

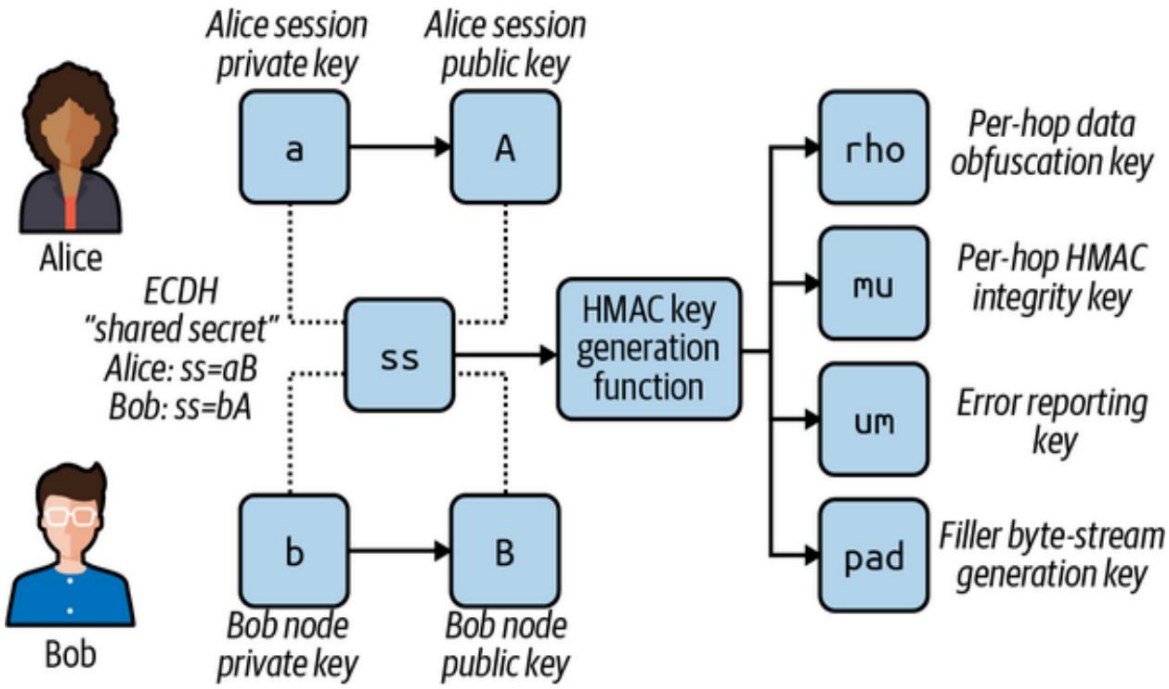

Figura 10-14. Generación de clave de cebolla

Clave de sesión de Alice

Para evitar revelar su identidad, Alice no usa la clave pública de su propio nodo para construir la cebolla. En su lugar, Alice crea una clave temporal de 32 bytes (256 bits) llamada ***clave privada de sesión*** y ***clave pública de sesión*** correspondiente .
Esto sirve como una "identidad" temporal y clave ***solo para esta cebolla.*** A partir de esta clave de sesión, Alice construirá todas las demás claves que se utilizarán en esta cebolla.

Detalles de generación de claves

La generación de claves, la generación de bytes aleatorios, las claves efímeras y cómo se utilizan en la construcción de paquetes se especifican en tres secciones de BOLT #4:

- Generación de claves
- Flujo de bytes aleatorios
- Construcción de paquetes

Por simplicidad y para evitar ser demasiado técnicos, no hemos incluido estos detalles en el libro. Consulte los enlaces anteriores si desea ver el

funcionamiento interno.

Generación de secretos compartidos Un

detalle importante que parece casi mágico es la capacidad de Alice de crear un ***secreto compartido*** con otro nodo simplemente conociendo sus claves públicas. Esto se basa en la invención del intercambio de claves (DH) Diffie-Hellman en la década de 1970 que revolucionó la criptografía. El enrutamiento de cebolla relámpago utiliza Elliptic Curve Diffie-Hellman (ECDH) en la curva secp256k1 de Bitcoin.

Es un truco tan genial que tratamos de explicarlo en términos simples en "Explicación de la curva elíptica Diffie-Hellman".

EXPLICACIÓN DE LA CURVA ELÍPTICA DIFFIE–HELLMAN

Suponga que la clave privada de Alice es ***a*** y la clave privada de Bob es b. Usando la curva elíptica, Alice y Bob multiplican cada uno su clave privada por el punto generador ***G*** para producir sus claves públicas ***A*** y B, respectivamente:

A = ***aG***

B = ***bG***

Ahora Alice y Bob pueden usar ***Elliptic Curve Diffie–Hellman Key Exchange*** para crear un ***ss secreto compartido,*** un valor que ambos pueden calcular de forma independiente sin intercambiar ninguna información

El secreto compartido ***ss*** se calcula por cada uno multiplicando su propia clave privada con la clave pública ***del otro ,*** de modo que:

ss = ***aB*** = ***bA***

Pero, ¿por qué estas dos multiplicaciones darían como resultado el mismo valor ***ss?*** Siga, mientras demostramos las matemáticas que prueban que esto es posible:

ss

= ***aB***

calculado por Alice que conoce tanto ***a*** (su clave privada) como ***B*** (la clave pública de Bob)

= ***a(bG)***

como sabemos que ***B*** = ***bG,*** sustituimos

= ***(ab)G***

debido a la asociatividad, podemos mover los paréntesis

= ***(ba) señor***

porque ***xy*** = ***yx*** (la curva es un grupo abeliano)

= ***b(ag)***

debido a la asociatividad, podemos mover los paréntesis

= ***bA***

y podemos sustituir ***aG*** por A.

El resultado ***bA*** puede ser calculado independientemente por Bob que conoce ***b*** (su clave privada) y ***A*** (la clave pública de Alice).

Por lo tanto, hemos demostrado que:

ss = ***aB*** (Alicia puede calcular esto)

ss = ***bA*** (Bob puede calcular esto)

Por lo tanto, cada uno puede calcular ***ss*** de forma independiente, que puede usar como clave compartida para cifrar simétricamente los secretos entre los dos sin comunicar el secreto compartido.

Una característica única de Sphinx como formato de paquete de red mixta es que, en lugar de incluir una clave de sesión distinta para cada salto en la ruta, lo que aumentaría drásticamente el tamaño del paquete de red mixta, se utiliza un esquema de ***cegamiento*** inteligente para aleatorizar de manera determinista la clave de sesión en cada salto.

En la práctica, este pequeño truco nos permite mantener el paquete de cebolla lo más compacto posible y al mismo tiempo conservar las propiedades de seguridad deseadas.

La clave de sesión para el salto i se deriva usando la clave pública del nodo y el secreto compartido derivado del salto i – 1:

session_key_i = session_key_{i-1} * SHA-256(node_pubkey_{i-1} || shared_secret_{i-1})

En otras palabras, tomamos la clave de sesión del salto anterior y la multiplicamos por un valor derivado de la clave pública y el secreto compartido derivado de ese salto.

Como la multiplicación de la curva elíptica se puede realizar en una clave pública sin conocer la clave privada, cada salto puede volver a aleatorizar la clave de sesión para el siguiente salto de manera determinista.

El creador del paquete cebolla conoce todos los secretos compartidos (ya que ha cifrado el paquete de forma única para cada salto) y, por lo tanto, puede derivar todos los factores de cegamiento.

Este conocimiento les permite derivar todas las claves de sesión utilizadas por adelantado durante la generación de paquetes.

Tenga en cuenta que el primer salto usa la clave de sesión original generada porque esta clave se usa para iniciar el cegamiento de la clave de sesión en cada salto subsiguiente.

Envolviendo las capas de cebolla

El proceso de envolver la cebolla se detalla en el TORNILLO n.º 4: Enrutamiento de cebolla, construcción de paquetes.

En esta sección describiremos este proceso en un nivel alto y algo simplificado, omitiendo ciertos detalles.

Cebollas De Longitud Fija

Hemos mencionado el hecho de que ninguno de los nodos de "salto" sabe cuánto dura la ruta o dónde se encuentran en la ruta. ¿Cómo es esto posible?

Si tiene un conjunto de direcciones, incluso si están encriptadas, ¿no puede saber qué tan lejos está del principio o del final simplemente mirando en ***qué*** parte de la lista de direcciones se encuentra?

El truco utilizado en el enrutamiento cebolla es hacer que la ruta (la lista de direcciones) siempre tenga la misma longitud para cada nodo. Esto se logra manteniendo el paquete de cebolla del mismo largo en cada paso.

En cada salto, la carga útil del salto aparece al comienzo de la carga útil de la cebolla, seguida de ***lo que parecen ser*** 19 cargas útiles más del salto. Cada salto se ve a sí mismo como el primero de 20 saltos.

PROPINA

La carga útil de la cebolla es de 1.300 bytes. Cada carga útil de salto es de 65 bytes o menos (rellenado a 65 bytes si es menos). Entonces, la carga útil total de la cebolla puede adaptarse a 20 cargas útiles de salto (1300 = 20 × 65). Por lo tanto, la ruta máxima enrutada de cebolla es de 20 saltos.

A medida que se "despega" cada capa, se agregan más datos de relleno (esencialmente basura) al final de la carga útil de la cebolla, de modo que el próximo salto obtenga una cebolla del mismo tamaño y sea nuevamente el "primer salto" en la cebolla.

El tamaño de la cebolla es de 1366 bytes, estructurado como se muestra en la figura 10-15:

1 byte

Un byte de versión

33 bytes

Una clave de sesión pública comprimida ("clave de sesión de Alice") a partir de la cual se puede generar el secreto compartido por salto ("Generación de secreto compartido") sin revelar la identidad de Alice

1300 bytes

La ***carga útil de cebolla*** real que contiene las instrucciones para cada salto

32 bytes

Una suma de comprobación de integridad HMAC

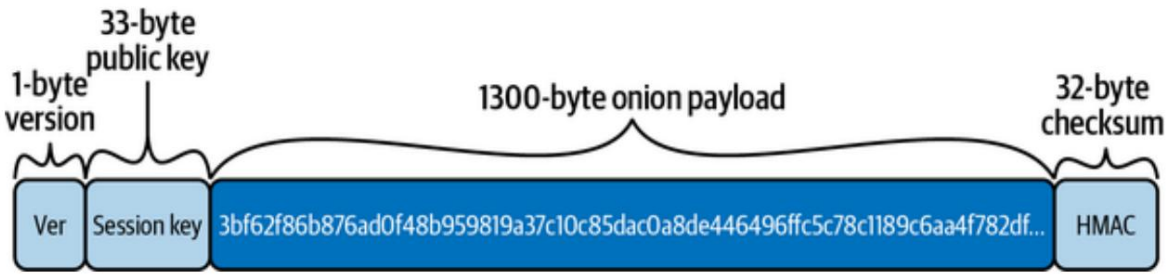

Figura 10-15. El paquete de cebolla

Un rasgo único de Sphinx como formato de paquete de red mixta es que, en lugar de incluir una clave de sesión distinta para cada salto en la ruta, lo que aumentaría drásticamente el tamaño del paquete de red mixta, en su lugar se utiliza un esquema de ***cegamiento*** inteligente para determinar de forma determinista aleatorizar la clave de sesión en cada salto.

En la práctica, este pequeño truco nos permite mantener el paquete de cebolla lo más compacto posible y al mismo tiempo conservar las propiedades de seguridad deseadas.

Envolviendo la cebolla (delineado)

Aquí está el proceso de envolver la cebolla, que se describe a continuación. Regrese a esta lista mientras exploramos cada paso con nuestro ejemplo del mundo real.

Para cada salto, el remitente (Alice) repite el mismo proceso:

1. Alice genera el secreto compartido por salto y el rho, mu y
   teclas de teclado
2. Alice genera 1300 bytes de relleno y llena el campo de carga útil de cebolla de 1300 bytes con este relleno.
3. Alice calcula el HMAC para la carga útil del salto (ceros para el final saltar).
4. Alice calcula la longitud de la carga útil del salto + HMAC + espacio para almacenar la longitud en sí.
5. Alice ***desplaza a la derecha*** la carga útil de la cebolla por el espacio calculado necesario para adaptarse a la carga útil del salto. Los datos de "relleno" más a la derecha se descartan, dejando suficiente espacio a la izquierda para la carga útil.

6. Alice inserta la carga útil de longitud + salto + HMAC al frente del campo de carga útil en el espacio creado al cambiar el relleno.
7. Alice usa la clave rho para generar un bloc de notas de un solo uso de 1300 bytes.
8. Alice ofusca toda la carga útil de la cebolla haciendo XORing con los bytes generados desde rho.
9. Alice calcula el HMAC de la carga útil de cebolla, utilizando el mu llave.
10. Alice agrega la clave pública de la sesión (para que el salto pueda calcular el secreto compartido).
11. Alice agrega el número de versión.
12. Alice vuelve a ocultar de manera determinista la clave de sesión usando un valor derivado al codificar el secreto compartido y la clave pública del salto anterior.

A continuación, Alice repite el proceso. Las nuevas claves se calculan, la carga útil de cebolla se desplaza (dejando caer más basura), la carga útil de nuevo salto se agrega al frente y toda la carga útil de cebolla se cifra con el flujo de bytes rho para el siguiente salto.

Para el salto final, el HMAC incluido en el Paso 3 sobre las instrucciones de texto sin formato es en realidad ***todo cero.*** El último salto usa esta señal para determinar que es efectivamente el último salto de la ruta. Alternativamente, también se puede usar el hecho de que el short_chan_id incluido en la carga útil para indicar el "siguiente salto" es todo cero.

Tenga en cuenta que en cada fase, la clave mu se usa para generar un HMAC sobre el paquete de cebolla ***cifrado*** (desde el punto de vista del nodo que procesa la carga útil), así como sobre el contenido del paquete con una sola capa de cifrado eliminada. Este HMAC externo permite que el nodo procese el paquete para verificar la integridad del paquete cebolla (sin modificar bytes). Luego, el HMAC interno se revela durante el proceso inverso de la rutina de "cambiar y cifrar" descrita anteriormente, que sirve como el HMAC ***externo*** para el siguiente salto.

Envolviendo la carga de lúpulo de Dina

Como recordatorio, la cebolla se envuelve comenzando al final del camino desde Dina, el nodo final o destinatario. Luego, la ruta se construye a la inversa hasta el remitente, Alice.

Alice comienza con un campo vacío de 1300 bytes, la ***carga útil de cebolla de longitud fija.*** Luego, llena la carga útil de la cebolla con un "relleno" de flujo de bytes pseudoaleatorio que se genera a partir de la clave del teclado.

Esto se muestra en la Figura 10-16.

> **NOTA**
>
> La generación de flujo de bytes aleatorios utiliza el algoritmo ChaCha20, como un generador criptográfico seguro de números pseudoaleatorios (CSPRNG). Dicho algoritmo generará un flujo determinista, largo y no repetitivo de bytes aparentemente aleatorios a partir de una semilla inicial. Los detalles se especifican en BOLT #4: Onion Routing, Pseudo Random Byte Stream.

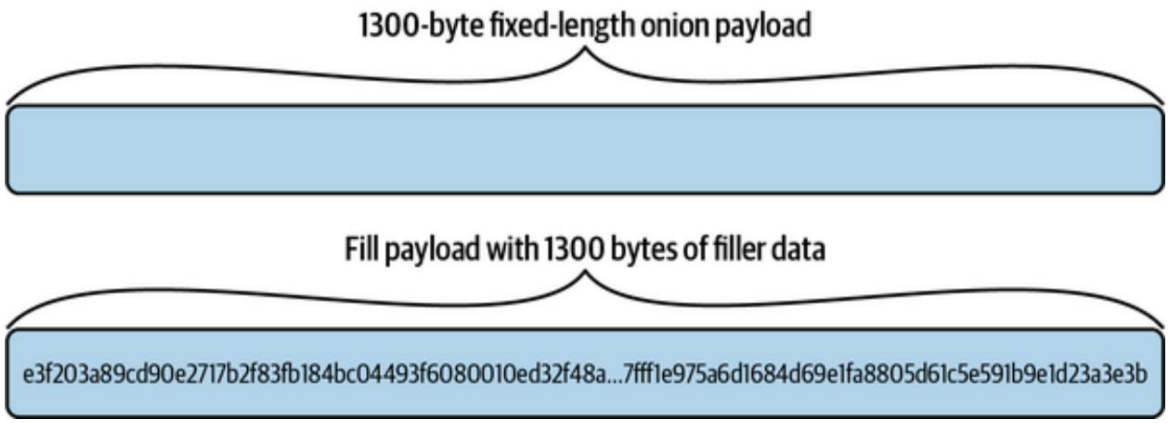

Figura 10-16. Llenar la carga útil de cebolla con un flujo de bytes aleatorio

Alice ahora insertará la carga útil de salto de Dina en el lado izquierdo de la matriz de 1300 bytes, desplazando el relleno hacia la derecha y descartando todo lo que se desborde. Esto se visualiza en la Figura 10-17.

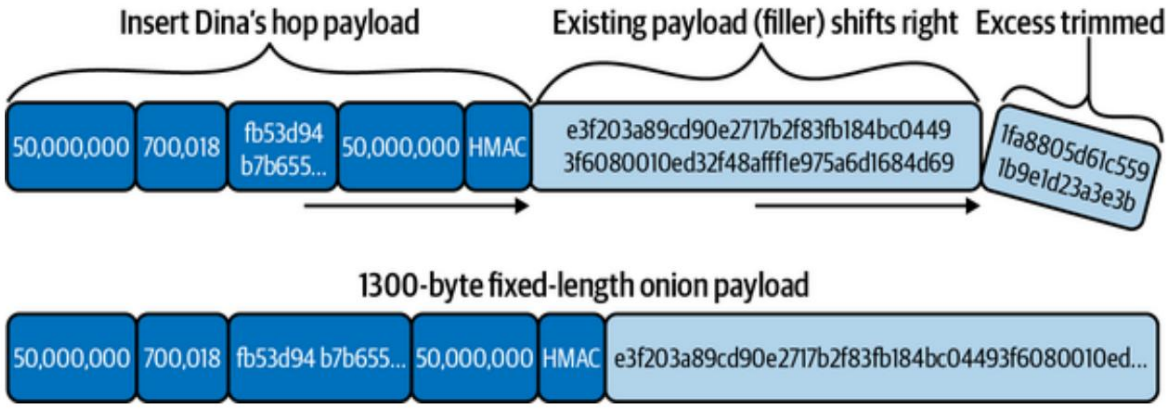

Figura 10-17. Agregar la carga útil de salto de Dina

Otra forma de ver esto es que Alice mide la longitud de la carga útil de salto de Dina, desplaza el relleno hacia la derecha para crear un espacio igual en el lado izquierdo de la carga útil de cebolla e inserta la carga útil de Dina en ese espacio.

En la siguiente fila hacia abajo, vemos el resultado: la carga útil de cebolla de 1300 bytes contiene La carga útil de salto de Dina y luego el flujo de bytes de relleno llenando el resto del espacio.

Luego, Alice ofusca toda la carga útil de la cebolla para que ***solo Dina*** pueda leerla.

Para hacer esto, Alice genera un flujo de bytes utilizando la clave rho (que Dina también conoce). Alice usa un bit a bit exclusivo o (XOR) entre los bits de la carga útil de cebolla y el flujo de bytes creado a partir de rho. El resultado aparece como un flujo de bytes aleatorio (o encriptado) de 1300 bytes de longitud. Este paso se muestra en la Figura 10-18.

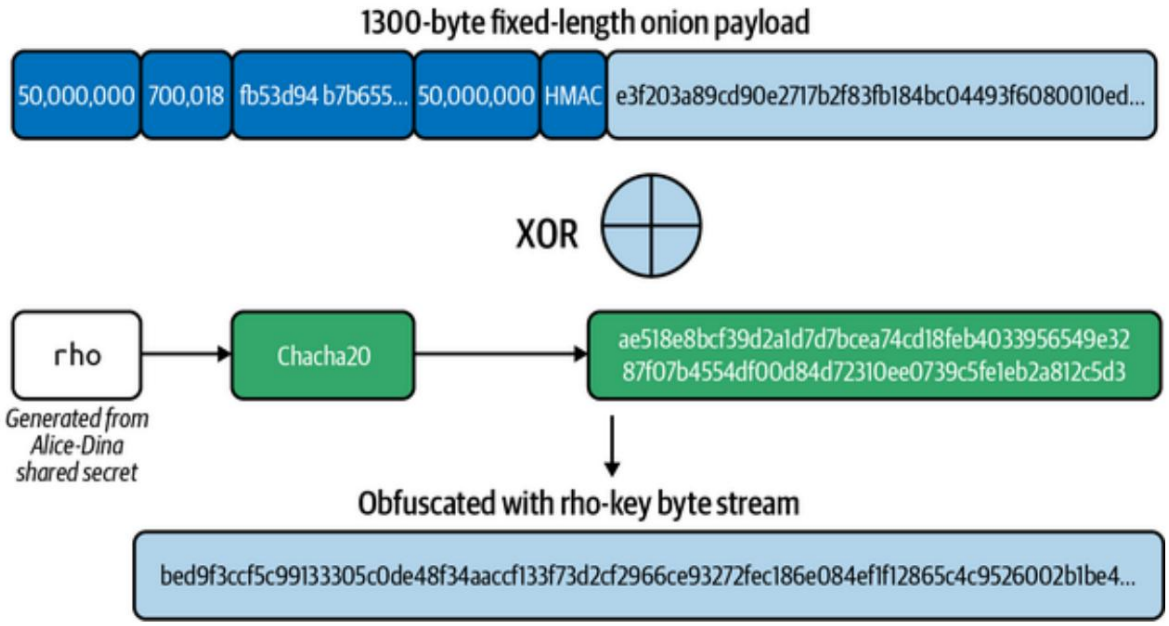

Figura 10-18. Ofuscando la carga útil de la cebolla

Una de las propiedades de XOR es que si lo hace dos veces, vuelve a los datos originales. Como veremos en "Bob De-Obfuscates His Hop Payload", si Dina aplica la misma operación XOR con el flujo de bytes generado desde rho, revelará el payload original de la cebolla.

PROPINA

XOR es una función ***involutiva ,*** lo que significa que si se aplica dos veces, se deshace. Específicamente XOR(XOR(a, b), b) = a. Esta propiedad se usa ampliamente en criptografía de clave simétrica.

Debido a que solo Alice y Dina tienen la clave rho (derivada del secreto compartido de Alice y Dina), solo ellas pueden hacer esto. Efectivamente, esto encripta la carga útil de la cebolla solo para los ojos de Dina.

Finalmente, Alice calcula un código de autenticación de mensajes basado en hash (HMAC) para la carga útil de Dina, que usa la clave mu como su clave de inicialización. Esto se muestra en la Figura 10-19.

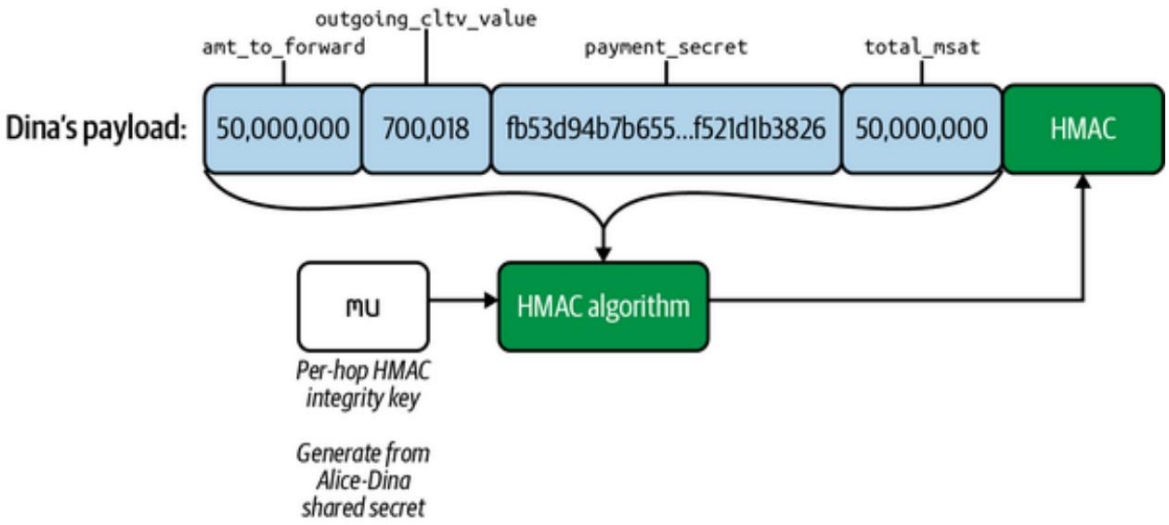

Figura 10-19. Agregar una suma de verificación de integridad HMAC a la carga útil de salto de Dina

Protección y detección de reproducción de enrutamiento de cebolla

El HMAC actúa como una suma de verificación segura y ayuda a Dina a verificar la integridad de la carga útil del salto. El HMAC de 32 bytes se agrega a la carga útil de salto de Dina. Tenga en cuenta que calculamos el HMAC sobre los datos ***cifrados*** en lugar de sobre los datos de texto sin formato. Esto se conoce como ***encrypt-then-mac*** y es la forma recomendada de usar un MAC, ya que proporciona integridad tanto de texto sin formato ***como*** de texto cifrado.

El cifrado autenticado moderno también permite el uso de un conjunto opcional de bytes de texto sin formato para autenticar también, conocido como ***datos asociados.*** En la práctica, esto suele ser algo así como un encabezado de paquete de texto sin formato u otra información auxiliar. Al incluir estos datos asociados en la carga útil que se va a autenticar (MAC'ed), el verificador de MAC garantiza que estos datos asociados no hayan sido manipulados (p. ej., intercambiando el encabezado de texto sin formato en un paquete cifrado).

En el contexto de Lightning Network, estos datos asociados se utilizan para ***fortalecer*** la protección de reproducción de este esquema. Como veremos a continuación, la protección de reproducción garantiza que un atacante no pueda ***retransmitir*** (reproducir) un paquete en la red y observar su ruta resultante. En su lugar, los nodos intermedios pueden utilizar las medidas de protección de reproducción definidas para detectar y rechazar un paquete reproducido. El formato base del paquete Sphinx utiliza un

registro de todas las claves secretas efímeras utilizadas para detectar repeticiones. Si alguna vez se vuelve a utilizar una clave secreta, el nodo puede detectarla y rechazar el paquete.

La naturaleza de los HTLC en Lightning Network nos permite fortalecer aún más la protección de reproducción al agregar un incentivo ***económico*** adicional. Recuerde que el hash de pago de un HTLC solo se puede usar de manera segura (para un pago completo) una vez. Si un hash de pago se usa nuevamente y atraviesa un nodo que ya ha visto el secreto de pago para ese hash, ¡entonces simplemente pueden retirar los fondos y cobrar el monto total del pago sin reenviar! Podemos usar este hecho para fortalecer la protección de reproducción al requerir que el ***hash de pago*** se incluya en nuestro cálculo HMAC como los datos asociados. Con este paso adicional, intentar reproducir un paquete de cebolla también requiere que el remitente se comprometa a usar el ***mismo*** hash de pago. Como resultado, además de la protección de reproducción normal, un atacante también puede perder la cantidad total del HTLC reproducido.

Una consideración con el conjunto cada vez mayor de claves de sesión almacenadas para la protección de reproducción es: ¿los nodos pueden reclamar este espacio? En el contexto de Lightning Network, la respuesta es: ¡sí! Una vez más, debido a los atributos únicos de la construcción HTLC, podemos realizar una mejora adicional sobre el protocolo básico de Sphinx. Dado que los HTLC son contratos ***de tiempo*** limitado basados en la altura absoluta del bloque, una vez que vence un HTLC, el contrato se cierra de manera efectiva y permanente. Como resultado, los nodos pueden usar esta altura de caducidad de CLTV (operador CHECKLOCKTIMEVERIFY) como un indicador para saber cuándo es seguro recolectar basura de una entrada en el registro anti-reproducción.

Envolviendo la carga de lúpulo de Chan

En la figura 10-20 vemos los pasos que se utilizan para envolver la carga útil de salto de Chan en la cebolla. Estos son los mismos pasos que Alice usó para envolver la carga útil de salto de Dina.

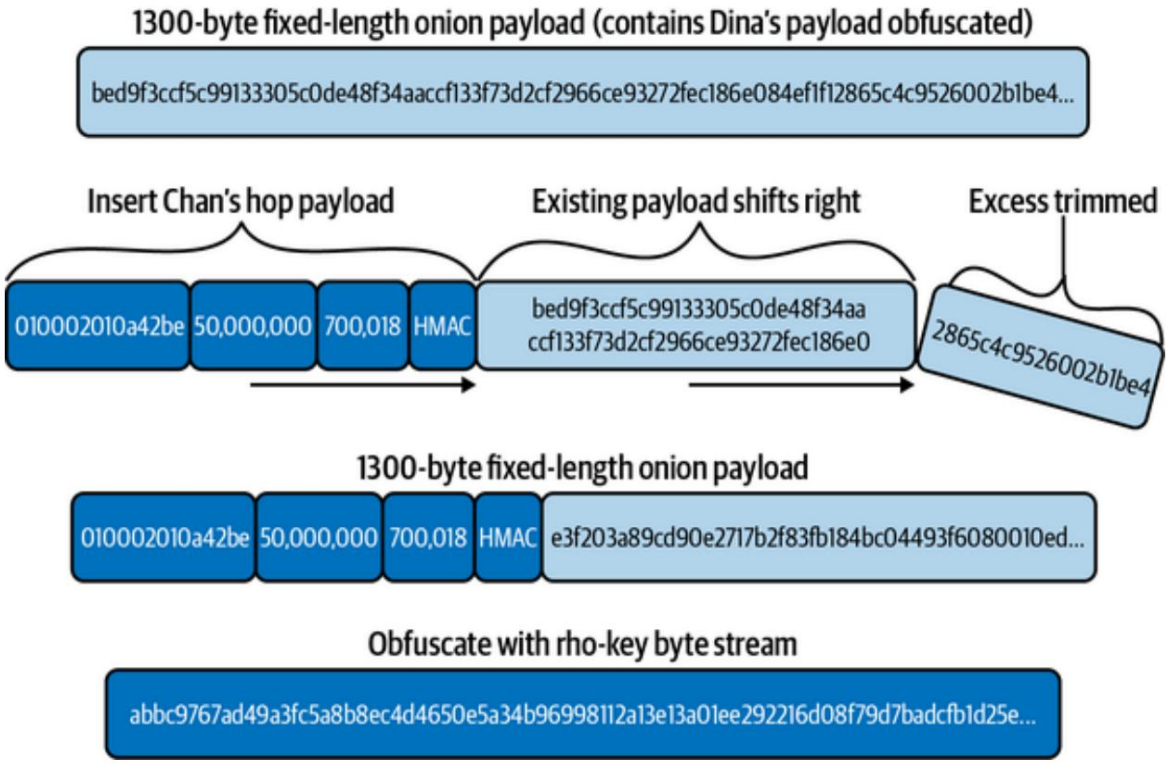

Figura 10-20. Envolviendo la cebolla para Chan

Alice comienza con la carga útil de 1300 cebollas creada para Dina. Los primeros 65 (o menos) bytes de esto son la carga útil de Dina ofuscada y el resto es relleno.
Alice debe tener cuidado de no sobrescribir la carga útil de Dina.

A continuación, Alice necesita ubicar la clave pública efímera (que se generó al comienzo de cada salto) que se agregará al paquete de enrutamiento en este salto.

Recuerde que en lugar de incluir una clave pública efímera única (que el remitente y el nodo intermedio usan en una operación ECDH para generar un secreto compartido), Sphinx usa una clave pública efímera única que se aleatoriza de forma determinista en cada salto.

Al procesar el paquete, Dina usará su secreto compartido y su clave pública para derivar el valor de cegamiento (b_dina) y lo usará para volver a aleatorizar la clave pública efímera, en una operación idéntica a la que realiza Alice durante la construcción inicial del paquete.

Alice agrega una suma de verificación HMAC interna a la carga útil de Chan y la inserta en el "frente" (lado izquierdo) de la carga útil de la cebolla, cambiando la carga útil existente al

derecho por una cantidad igual. Recuerde que efectivamente se utilizan ***dos*** HMAC en el esquema: el HMAC externo y el HMAC interno. En este caso, el HMAC ***interno*** de Chan es en realidad el HMAC ***externo*** de Dina .

Ahora la carga útil de Chan está al frente de la cebolla. Cuando Chan ve esto, no tiene idea de cuántas cargas útiles vinieron antes o después. ¡Parece el primero de 20 saltos siempre!

A continuación, Alice ofusca toda la carga útil mediante XOR con el flujo de bytes generado a partir de la clave rho de Alice-Chan. Solo Alice y Chan tienen esta clave rho, y solo ellos pueden producir el flujo de bytes para ofuscar y desofuscar la cebolla. Finalmente, como hicimos en el paso anterior, calculamos el HMAC externo de Chan, que es lo que usará para verificar la integridad del paquete de cebolla encriptado.

Envolviendo la carga de lúpulo de Bob

En la figura 10-21 vemos los pasos que se utilizan para envolver la carga útil de lúpulo de Bob en la cebolla.

¡Muy bien, ahora esto es fácil!

Comience con la carga de cebolla (ofuscada) que contiene las cargas de lúpulo de Chan y Dina.

Obtenga la clave de sesión para este salto derivada del factor de cegamiento generado por el salto anterior. Incluya el HMAC externo del salto anterior como el HMAC interno de este salto. Inserte la carga útil de lúpulo de Bob al principio y cambie todo lo demás hacia la derecha, dejando caer un fragmento del tamaño de la carga útil de Bob-hop desde el final (de todos modos, era relleno).

Ofusque todo el asunto XOR con la clave rho del secreto compartido de Alice-Bob para que solo Bob pueda desenvolverlo.

Calcule el HMAC externo y péguelo al final de la carga útil del salto de Bob.

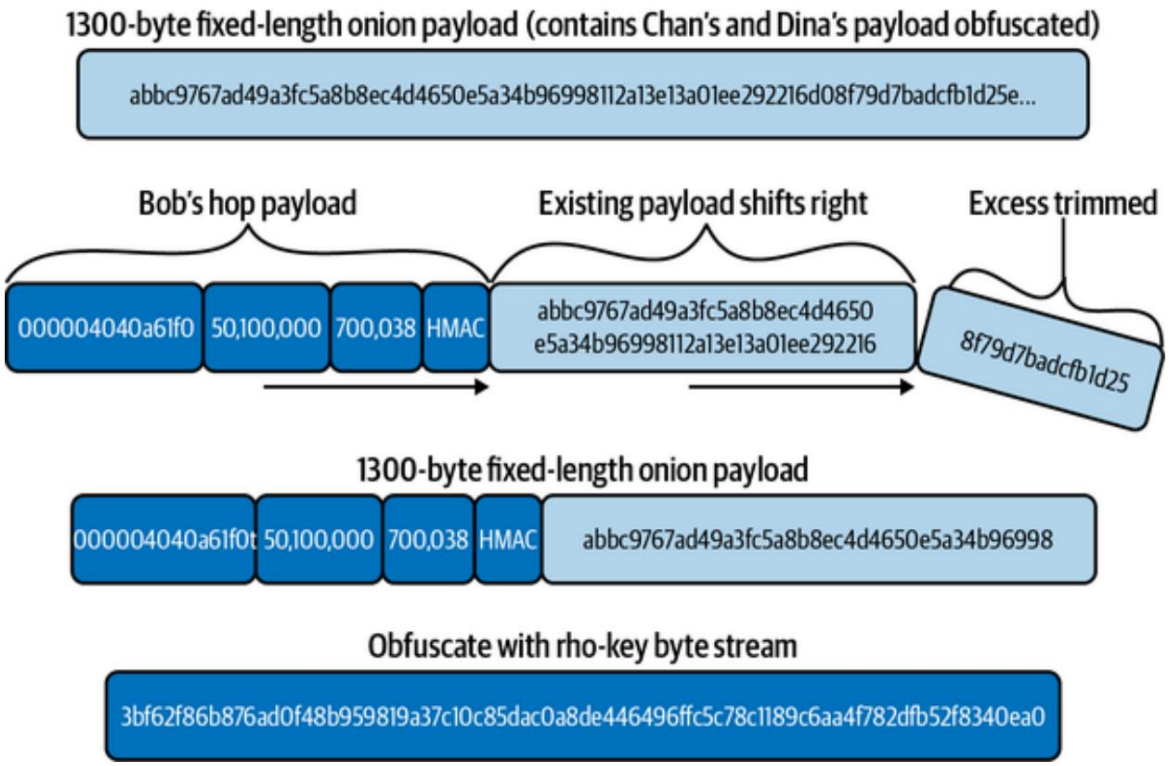

Figura 10-21. Envolviendo la cebolla para Bob

El último paquete de cebolla

La carga útil final de la cebolla está lista para enviarse a Bob. Alice no necesita agregar más cargas útiles de salto.

Alice calcula un HMAC para la carga útil de la cebolla para asegurarla criptográficamente con una suma de verificación que Bob puede verificar.

Alice agrega una clave de sesión pública de 33 bytes que utilizará cada salto para generar un secreto compartido y las claves rho, mu y pad.

Finalmente, Alice pone el número de versión de cebolla (0 actualmente) al frente. Esto permite futuras actualizaciones del formato del paquete de cebolla.

El resultado se puede ver en la Figura 10-22.

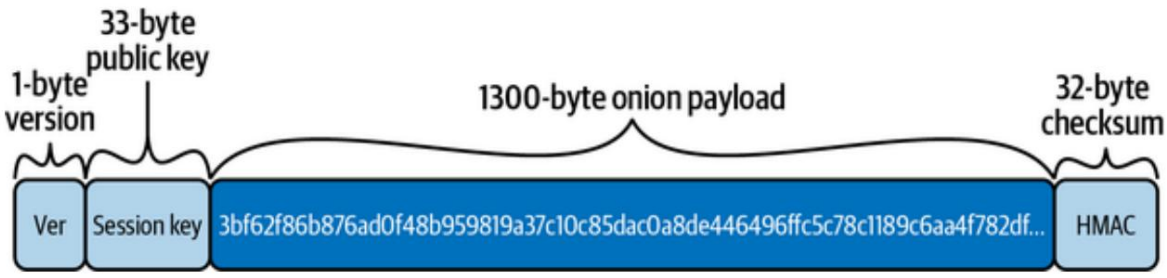

Figura 10-22. El paquete de cebolla

Enviando la cebolla

En esta sección, veremos cómo se reenvía el paquete cebolla y cómo se implementan los HTLC a lo largo de la ruta.

Los paquetes Onion del mensaje

update_add_htlc se envían como parte del mensaje update_add_htlc. Si recuerda del "Mensaje update_add_HTLC", en el Capítulo 9, vimos que el contenido del mensaje update_add_htlc es el siguiente:

```
[channel_id:channel_id] [u64:id]
[u64:amount_msat]
[sha256:pago_hash]
[u32:cltv_expiry]
[1366*byte:onion_routing_packet]
```

Recordará que este mensaje lo envía un socio de canal para pedirle al otro socio de canal que agregue un HTLC. Así es como Alice le pedirá a Bob que agregue un HTLC para pagarle a Dina. Ahora comprende el propósito del último campo,ion_routing_packet, que tiene una longitud de 1366 bytes. ¡Es el paquete de cebolla completamente envuelto que acabamos de construir!

Alice envía la cebolla a Bob

Alice enviará el mensaje update_add_htlc a Bob. Veamos qué contendrá este mensaje:

Canal ID

Este campo contiene el ID del canal Alice-Bob, que en nuestro ejemplo es 0000031e192ca1 (consulte la Figura 10-9).

identificación

El ID de este HTLC en este canal, a partir de 0.

cantidad_msat

El monto del HTLC: 50,200,000 millisatoshis.

pago_hash

El hash de pago RIPEMD160(SHA-256):

9e017f6767971ed7cea17f98528d5f5c0ccb2c71.

cltv_expiry

El límite de tiempo de vencimiento para el HTLC será 700,058. Alice agrega 20 bloques al conjunto de caducidad en la carga útil de Bob de acuerdo con cltv_expiry_delta negociado por Bob.

paquete_enrutamiento_cebolla

¡El último paquete de cebolla que Alice construyó con todas las cargas útiles de lúpulo!

Bob revisa la cebolla

Como vimos en el Capítulo 9, Bob agregará el HTLC a las transacciones de compromiso y actualizará el estado del canal con Alice.

Bob desenvolverá la cebolla que recibió de Alice de la siguiente manera:

1. Bob toma la clave de sesión del paquete de cebolla y obtiene la Alice-Bob compartió el secreto.

2. Bob genera la clave mu a partir del secreto compartido y la usa para
   verifique la suma de verificación HMAC del paquete de cebolla.

Ahora que Bob generó la clave compartida y verificó el HMAC, puede comenzar a desenvolver la carga útil de cebolla de 1300 bytes dentro del paquete de cebolla. El objetivo es que Bob recupere su propia carga útil de salto y luego envíe la cebolla restante al siguiente salto.

Si Bob extrae y elimina su carga útil de lúpulo, la cebolla restante no será de 1300 bytes, ¡será más corta! Entonces, el próximo salto sabrá que no es el primer salto y podrá detectar qué tan largo es el camino. Para evitar esto, Bob necesita agregar más relleno para rellenar la cebolla.

Bob genera relleno

Bob genera relleno de una manera ligeramente diferente a Alice, pero siguiendo el mismo principio general.

Primero, Bob ***amplía*** la carga útil de la cebolla en 1300 bytes y los completa con valores 0. Ahora el paquete cebolla tiene una longitud de 2600 bytes, la primera mitad contiene los datos que envió Alice y la siguiente mitad contiene ceros. Esta operación se muestra en la Figura 10-23.

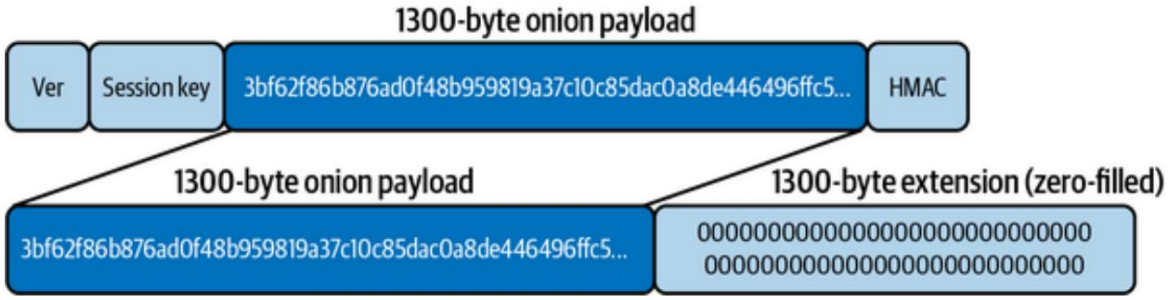

Figura 10-23. Bob amplía la carga útil de la cebolla en 1300 bytes (llenos con ceros)

Este espacio vacío se ofuscará y se convertirá en "relleno" mediante el mismo proceso que utiliza Bob para des-ofuscar su propia carga útil de lúpulo. Veamos cómo funciona eso.

Bob desofusca su carga de lúpulo

A continuación, Bob generará la clave rho a partir de la clave compartida Alice-Bob. Usará esto para generar un flujo de 2600 bytes, utilizando el algoritmo ChaCha20.

PROPINA

Los primeros 1300 bytes del flujo de bytes generado por Bob son exactamente los mismos que los generados por Alice usando la clave rho.

A continuación, Bob aplica los 2600 bytes del flujo de bytes rho a la carga útil de cebolla de 2600 bytes con una operación XOR bit a bit.

Los primeros 1300 bytes quedarán desofuscados por esta operación XOR, porque es la misma operación que aplicó Alice y XOR es involutivo. Entonces Bob ***revelará*** su carga útil de salto seguida de algunos datos que parecen codificados.

Al mismo tiempo, aplicar el flujo de bytes rho a los 1300 ceros que se agregaron a la carga de cebolla los convertirá en datos de relleno aparentemente aleatorios. Esta operación se muestra en la Figura 10-24.

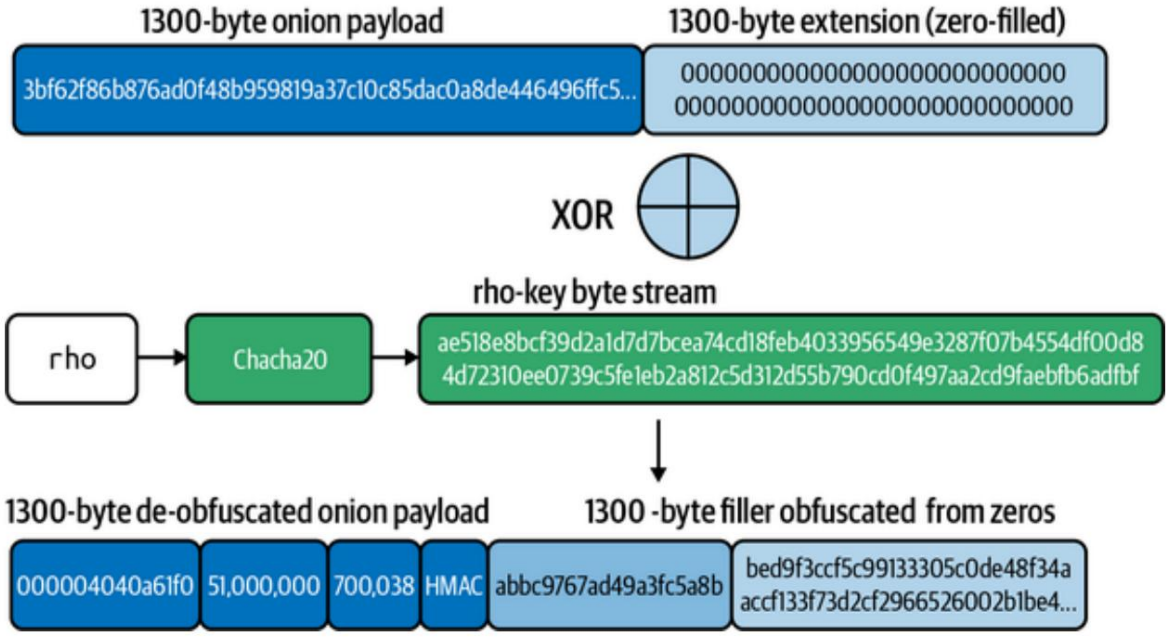

Figura 10-24. Bob desofusca la cebolla, ofusca el relleno

Bob extrae el HMAC externo para el próximo salto

Recuerde que se incluye un HMAC interno para cada salto, que luego se convertirá en el HMAC externo para el ***siguiente*** salto. En este caso, Bob extrae el HMAC interno (ya verificó la integridad del paquete encriptado con el HMAC externo) y lo deja a un lado porque lo agregará al paquete ofuscado para permitir que Chan verifique el HMAC de su paquete encriptado. paquete.

Bob quita su carga útil y desplaza la cebolla a la izquierda

Ahora Bob puede eliminar su carga útil de lúpulo del frente de la cebolla y desplazar a la izquierda los datos restantes. Una cantidad de datos igual a la carga útil de salto de Bob de los 1300 bytes de relleno de la segunda mitad ahora se trasladará al espacio de carga útil de cebolla. Esto se muestra en la Figura 10-25.

Ahora Bob puede quedarse con la primera mitad de 1300 bytes y descartar los 1300 bytes extendidos (de relleno).

Bob ahora tiene un paquete cebolla de 1300 bytes para enviar al siguiente salto. Es casi idéntico al payload de cebolla que Alice había creado para Chan, excepto que Bob colocó los últimos 65 bytes de relleno y serán diferentes.

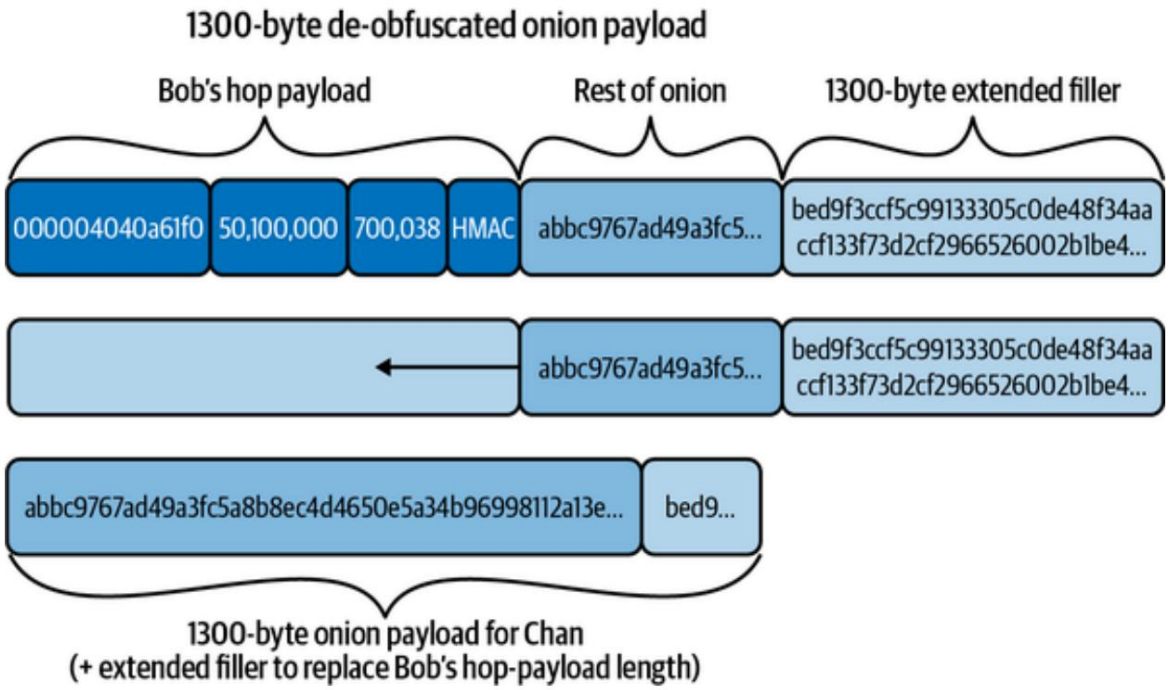

Figura 10-25. Bob elimina la carga útil del lúpulo y desplaza el resto a la izquierda, llenando el espacio con un nuevo relleno

Nadie puede notar la diferencia entre el relleno puesto por Alice y el relleno puesto por Bob. ¡El relleno es relleno! Todos son bytes aleatorios de todos modos. Tenga en cuenta que si Bob (o uno de los otros alias de Bob) está presente en la ruta en dos ubicaciones distintas, entonces pueden notar la diferencia porque el protocolo base siempre usa el mismo hash de pago en toda la ruta. Los pagos multitrayecto atómicos (AMP) y los contratos de bloqueo de tiempo puntual (PTLC) eliminan el vector de correlación al aleatorizar el identificador de pago en cada ruta/salto.

Bob construye el nuevo paquete de cebolla

Bob ahora copia la carga útil de cebolla en el paquete de cebolla, agrega el HMAC externo para chan, vuelve a aleatorizar la clave de sesión (de la misma manera que lo hace Alice, el remitente) con la operación de multiplicación de curva elíptica y agrega un byte de versión nueva.

Para volver a aleatorizar la clave de sesión, Bob primero calcula el factor de cegamiento de su salto, utilizando su clave pública de nodo y el secreto compartido que derivó:

```
b_bob = SHA-256(P_bob || shared_secret_bob)
```

Con esto generado, Bob ahora vuelve a aleatorizar la clave de sesión realizando una multiplicación de EC usando su clave de sesión y el factor de cegamiento:

session_key_chan = session_key_bob * b_bob

La clave pública session_key_chan se agregará al frente del paquete de cebolla para que Chan la procese.

Bob verifica los detalles de HTLC

La carga útil de salto de Bob contiene las instrucciones necesarias para crear un HTLC para Chan.

En la carga útil del salto, Bob encuentra short_channel_id, amt_to_forward y cltv_expiry.

Primero, Bob verifica si tiene un canal con esa identificación corta. Descubre que tiene ese canal con Chan.

Luego, Bob confirma que el monto saliente (50 100 satoshis) es menor que el monto entrante (50 200 satoshis) y, por lo tanto, se cumplen las expectativas de honorarios de Bob.

De manera similar, Bob verifica que el cltv_expiry saliente sea menor que el cltv_expiry entrante, lo que le da a Bob suficiente tiempo para reclamar el HTLC entrante si hay una infracción.

Bob envía el update_add_htlc a Chan

Bob ahora construye y HTLC para enviar a Chan, de la siguiente manera:

Canal ID

Este campo contiene el ID del canal Bob-Chan, que en nuestro ejemplo es 000004040a61f0 (consulte la Figura 10-9).

identificación

El ID de este HTLC en este canal, a partir de 0.

cantidad_msat

El monto del HTLC: 50,100,000 millisatoshis.

pago_hash

El hash de pago RIPEMD160(SHA-256):
9e017f6767971ed7cea17f98528d5f5c0 ccb2c71.

Esto es lo mismo que el hash de pago del HTLC de Alice.

cltv_expiry

El límite de tiempo de vencimiento para el HTLC será 700,038.

paquete_enrutamiento_cebolla

El paquete de cebollas que Bob reconstruyó después de retirar su carga útil de lúpulo.

Chan reenvía la cebolla

Chan repite exactamente el mismo proceso que Bob:

1. Chan recibe el update_add_htlc y procesa el HTLC
   solicitud, agregándolo a las transacciones de compromiso.
2. Chan genera la clave compartida Alice-Chan y la subclave mu.
3. Chan verifica el paquete de cebolla HMAC, luego extrae la carga útil de cebolla de 1300 bytes.
4. Chan amplía la carga útil de cebolla en 1300 bytes adicionales, llenándolo con ceros
5. Chan usa la tecla rho para producir 2600 bytes.
6. Chan usa el flujo de bytes generado para XOR y desofusca la carga útil de la cebolla. Simultáneamente, la operación XOR ofusca los 1300 ceros adicionales, convirtiéndolos en relleno.

7. Chan extrae el HMAC interno en la carga útil, que se convertirá el HMAC externo para Dina.

8. Chan elimina su carga útil de salto y desplaza la carga útil de cebolla a la izquierda en la misma cantidad. Parte del relleno generado en los 1300 bytes extendidos se traslada a los 1300 bytes de la primera mitad, convirtiéndose en parte de la carga útil de la cebolla.

9. Chan construye el paquete de cebolla para Dina con esta carga útil de cebolla.

10. Chan crea un mensaje update_add_htlc para Dina y inserta el paquete de cebolla en él.

11. Chan envía el update_add_htlc a Dina.

12. Chan vuelve a aleatorizar la clave de sesión como lo hizo Bob en el salto anterior para En.

Dina recibe la carga útil final

Cuando Dina recibe el mensaje update_add_htlc de Chan, sabe por el pago_hash que se trata de un pago para ella. Ella sabe que es el último salto en la cebolla.

Dina sigue exactamente el mismo proceso que Bob y Chan para verificar y desenvolver la cebolla, excepto que no construye un relleno nuevo ni envía nada. En cambio, Dina le responde a Chan con update_fulfill_htlc para canjear el HTLC. El update_fulfill_htlc fluirá hacia atrás a lo largo de la ruta hasta que llegue a Alice. Todos los HTLC se canjean y los saldos de los canales se actualizan. ¡El pago está completo!

Devolviendo errores

Hasta ahora, hemos analizado la propagación hacia adelante de la cebolla que establece los HTLC y la propagación hacia atrás del secreto de pago que deshace los HTLC una vez que el pago es exitoso.

Hay otra función muy importante del enrutamiento cebolla: ***retorno de error.*** Si hay un problema con el pago, la cebolla o los saltos, debemos propagar un error hacia atrás para informar a todos los nodos de la falla y deshacer cualquier HTLC.

Los errores generalmente se dividen en tres categorías: fallas de cebolla, fallas de nodo y fallas de canal. Estos, además, pueden subdividirse en errores permanentes y transitorios. Finalmente, algunos errores contienen actualizaciones de canales para ayudar con futuros intentos de entrega de pagos.

> **NOTA**
>
> A diferencia de los mensajes en el protocolo peer-to-peer (P2P) (definido en BOLT #2: Peer Protocol for Channel Management), los errores no se envían como mensajes P2P sino que se envuelven dentro de paquetes de retorno de cebolla y siguen el camino inverso de la cebolla (retropropagación).

La devolución de errores se define en el PERNO n.° 4: Enrutamiento cebolla, devolución de errores.

Los errores son codificados por el nodo de retorno (el que descubrió un error) en un ***paquete de retorno*** de la siguiente manera:

```
[32*byte:hmac]
[u16:failure_len]
[failure_len*byte:failuremsg] [u16:pad_len]
[pad_len*byte:pad]
```

La suma de verificación de verificación HMAC del paquete de retorno se calcula con la clave um, generada a partir del secreto compartido establecido por la cebolla.

> PROPINA
>
> El nombre de la clave um es el reverso del nombre mu, lo que indica el mismo uso pero en la dirección opuesta (propagación hacia atrás).

A continuación, el nodo de retorno genera una clave ammag (inversa de la palabra "gamma") y ofusca el paquete de retorno mediante una operación XOR

con un flujo de bytes generado desde ammag.

Finalmente, el nodo de retorno envía el paquete de retorno al salto del que recibió la cebolla original.

Cada salto que recibe un error generará una clave ammag y ofuscará el paquete de retorno nuevamente mediante una operación XOR con el flujo de bytes de moho.

Eventualmente, el remitente (nodo de origen) recibe un paquete de retorno. Luego generará claves ammag y um para cada salto y XOR desofuscará el error de retorno de forma iterativa hasta que revele el paquete de retorno.

Mensajes de error

El mensaje de error se define en el PERNO n.º 4: Enrutamiento cebolla, mensajes de error.

Un mensaje de falla consta de un código de falla de dos bytes seguido de los datos aplicables a ese tipo de falla.

El byte superior del código_fallo es un conjunto de indicadores binarios que se pueden combinar (con OR binario):

0x8000 *(BADONION)*

Cebolla no analizable cifrada mediante el envío de pares

0x4000 *(PERMANENTE)*

Fallo permanente (de lo contrario, transitorio)

0x2000 *(NODO)*

Fallo de nodo (de lo contrario, canal)

0x1000 *(ACTUALIZAR)*

Nueva actualización de canal adjunta

Los tipos de falla que se muestran en la Tabla 10-1 están definidos actualmente.

y

s

Escribe	Nombre simbólico	Significado
PERM|1	reino_inválido	El nodo de procesamiento no entendió el byte del reino
NODO|2	falla_de_nodo_temporal ure	Fallo temporal general del nodo de procesamiento
PERM|NODO|2	falla_nodo_permanente ure	Fallo permanente general del nodo de procesamiento
PERM|NODE|3	required_node_fea tu re_missing	El nodo de procesamiento tiene una característica requerida que no estaba en esta cebolla
BADONIÓN|PERMANENTE |4	invalid_onion_version en	El byte de versión no fue entendido por el nodo de procesamiento
BADONIÓN|PERMANENTE |5	invalid_onion_hmac	El HMAC de la cebolla era incorrecto cuando llegó al nodo de procesamiento
BADONIÓN|PERMANENTE |6	invalid_onion_key	El nodo de procesamiento no pudo analizar la clave efímera
ACTUALIZAR|7	canal_temporal_fallo	El canal del nodo de procesamiento no pudo manejar este HTLC, pero es posible que pueda manejarlo, u otros, más adelante
PERM|8	falla_del_canal_permanente	El canal del nodo de procesamiento no puede manejar ningún HTLC
PERM|9	canal_requerido_característica_perdida	El canal del nodo de procesamiento requiere características que no están presentes en la cebolla.
PERM|10	unknown_next_peer	La cebolla especificó un short_channel_id que no coincide con ningún líder del nodo de procesamiento
ACTUALIZAR|11	cantidad_debajo_minimu	La cantidad de HTLC estaba por debajo de htlc_min

	metro	imum_msat del canal de el nodo de procesamiento
ACTUALIZAR|12	tarifa_insuficiente	El monto de la tarifa fue inferior al requerido por la canal desde el nodo de procesamiento
ACTUALIZAR|13	incorrecta_cltv_expi ry	El cltv_expiry no cumple con el cltv_expiry_delta requerido por el canal desde el nodo de procesamiento
ACTUALIZAR|14	caducidad_demasiado_pronto	El vencimiento de CLTV está demasiado cerca del actual altura del bloque para seguridad manejo por el nodo de procesamiento
PERM|15	incor recto_o_desconocido n_ detalles_de_pago	El pago_hash es desconocido para el final nodo, el payment_secret no hacer coincidir el pago_hash, la cantidad para que payment_hash es incorrecto, o el vencimiento CLTV del HTLC está demasiado cerca de la altura del bloque actual para seguridad manejo
18	final_incor rect_ clt v_caducidad	El vencimiento de CLTV en el HTLC no coincide el valor de la cebolla
19	final_incor rect_htl c_cantidad	La cantidad en el HTLC no coincide con la valor en la cebolla
ACTUALIZAR|20	canal_deshabilitado	El canal del nodo de procesamiento tiene sido deshabilitado
21	vencimiento_demasiado_lejos	El vencimiento de CLTV en el HTLC está demasiado lejos en el futuro
PERM|22	invalid_onion_paylo anuncio	La carga útil de cebolla por salto descifrada no fue entendido por el nodo de procesamiento o esta incompleto
23	mpp_timeout	El monto total del pago de varias partes no fue recibido dentro de un tiempo razonable

Pagos atascados

En la implementación actual de Lightning Network, existe la posibilidad de que un intento de pago se ***atasque:*** ni realizado ni cancelado por un error. Esto puede suceder debido a un error en un nodo intermediario, un nodo que se desconecta mientras maneja los HTLC o un nodo malicioso que contiene los HTLC sin informar un error. En todos estos casos, el HTLC no se puede resolver hasta que expire. El bloqueo de tiempo (CLTV) que se establece en cada HTLC ayuda a resolver esta condición (entre otras posibles fallas de canal y enrutamiento de HTLC).

Sin embargo, esto significa que el remitente del HTLC tiene que esperar hasta el vencimiento, y los fondos comprometidos con ese HTLC no estarán disponibles hasta que venza el HTLC.

Además, el remitente ***no puede volver a intentar*** el mismo pago porque, si lo hace, corre el riesgo de que ***tanto*** el pago original como el que se ha vuelto a intentar tengan éxito: al destinatario se le paga dos veces. Esto se debe a que, una vez enviado, el remitente no puede "cancelar" un HTLC; tiene que fallar o caducar.
Los pagos atascados, aunque son raros, crean una experiencia de usuario no deseada, donde la billetera del usuario no puede pagar o cancelar un pago.

Una solución propuesta a este problema se llama ***pagos atascados,*** y depende de los contratos de bloqueo de tiempo de punto (PTLC), que son contratos de pago que utilizan una primitiva criptográfica diferente a los HTLC (es decir, suma de puntos en la curva elíptica en lugar de un hash y preimagen secreta).
Los PTLC son engorrosos con ECDSA, pero mucho más fáciles con las funciones de firma Taproot y Schnorr de Bitcoin, que se bloquearon recientemente para su activación en noviembre de 2021. Se espera que los PTLC se implementen en Lightning Network después de que se activen estas funciones de Bitcoin.

Pagos espontáneos Keysend

En el flujo de pago descrito anteriormente en el capítulo, asumimos que Dina recibió una factura de Alice "fuera de banda", o la obtuvo a través de algún mecanismo no relacionado con el protocolo (normalmente copiar/pegar o escaneos de códigos QR). Esta característica significa que el proceso de pago siempre toma dos pasos:

primero, el remitente obtiene una factura y, segundo, usa el hash de pago (codificado en la factura) para enrutar correctamente un HTLC. El viaje de ida y vuelta adicional requerido para obtener una factura antes de realizar un pago puede ser un cuello de botella en las aplicaciones que implican la transmisión de micropagos a través de Lightning. ¿Qué pasaría si pudiéramos simplemente "pasar" un pago espontáneamente, sin tener que obtener primero una factura del destinatario? El protocolo de envío de claves es una extensión de extremo a extremo (solo el remitente y el receptor son conscientes) del protocolo Lightning que permite pagos automáticos espontáneos.

Registros TLV de cebolla personalizados

El protocolo Lightning moderno utiliza la codificación TLV (Tipo-Longitud-Valor) en la cebolla para codificar la información que le dice a cada nodo ***dónde*** y ***cómo*** reenviar el pago. Aprovechando el formato TLV, a cada parte de la información de enrutamiento (como el siguiente nodo al que pasar el HTLC) se le asigna un tipo específico (o clave) codificado como un número entero de longitud variable BigSize (tamaño máximo como número entero de 64 bits). Estos tipos "esenciales" (valores invertidos por debajo de 65536) se definen en el BOLT #4, junto con el resto de los detalles de enrutamiento de cebolla. Los tipos de cebolla con un valor superior a 65536 están destinados a ser utilizados por billeteras y aplicaciones como "registros personalizados".

Los registros personalizados permiten que las aplicaciones de pago adjunten metadatos o contexto adicionales a un pago como pares clave/valor en la cebolla. Dado que los registros personalizados se incluyen en la propia carga útil de la cebolla, como todos los demás contenidos de salto, los registros se cifran de extremo a extremo. Como los registros personalizados consumen efectivamente una parte del paquete de cebolla de 1300 bytes de tamaño fijo, la codificación de cada clave y valor de cada registro personalizado reduce la cantidad de espacio disponible para codificar el resto de la ruta. En la práctica, esto significa que cuanto más espacio de cebolla se utilice para los registros personalizados, más corta puede ser la ruta. Dado que cada paquete HTLC tiene un tamaño fijo, los registros personalizados no ***agregan*** ningún dato adicional a un HTLC; más bien, reasignan bytes que de otro modo se habrían llenado con datos aleatorios.

Envío y recepción de pagos Keysend

Un pago de envío de clave invierte el flujo típico de un HTLC donde el receptor revela una preimagen secreta al remitente. En cambio, el remitente incluye la preimagen ***dentro*** de la cebolla al receptor y enruta el HTLC al receptor. Luego, el receptor descifra la carga útil de la cebolla y utiliza la preimagen incluida (que ***debe*** coincidir con el hash de pago del HTLC) para liquidar el pago. Como resultado, los pagos de envío de llaves se pueden realizar sin obtener primero una factura del receptor, ya que la preimagen se "empuja" hacia el receptor. Un pago de envío de clave utiliza un tipo de registro personalizado TLV de 5482373484 para codificar un valor de preimagen de 32 bytes.

Keysend y registros personalizados en aplicaciones Lightning

Muchas aplicaciones Lightning de transmisión utilizan el protocolo keyend para transmitir continuamente satoshis a un destino identificado por su clave pública en la red. Por lo general, una aplicación también incluirá metadatos, como una nota de propina/donación u otra información a nivel de la aplicación, además del registro de envío de llaves.

Conclusión

El protocolo de enrutamiento de cebolla de Lightning Network está adaptado del protocolo Sphinx para satisfacer mejor las necesidades de una red de pago. Como tal, ofrece una gran mejora en la privacidad y la contravigilancia en comparación con la cadena de bloques pública y transparente de Bitcoin.

En el Capítulo 12 , veremos cómo Alice utiliza la combinación de enrutamiento de origen y enrutamiento de cebolla para encontrar una buena ruta y enrutar el pago a Dina. Para encontrar una ruta, Alice primero debe conocer la topología de la red, que es el tema del Capítulo 11.

---

1 George Danezis e Ian Goldberg, “Sphinx: A Compact and Provably Secure Mix Format”, en ***Simposio IEEE sobre seguridad y privacidad*** (Nueva York: IEEE, 2009), 269–282.

Capítulo 11. El chisme y el gráfico de canal

En este capítulo, describiremos el protocolo de chismes de Lightning Network y cómo los nodos lo utilizan para construir y mantener un gráfico de canal. También revisaremos el mecanismo de arranque de DNS que se utiliza para encontrar compañeros con los que "chismear".

La sección "Tarifas de enrutamiento y retransmisión de chismes" está resaltada por un esquema que abarca la capa de enrutamiento y la capa de igual a igual de la Figura 11-1.

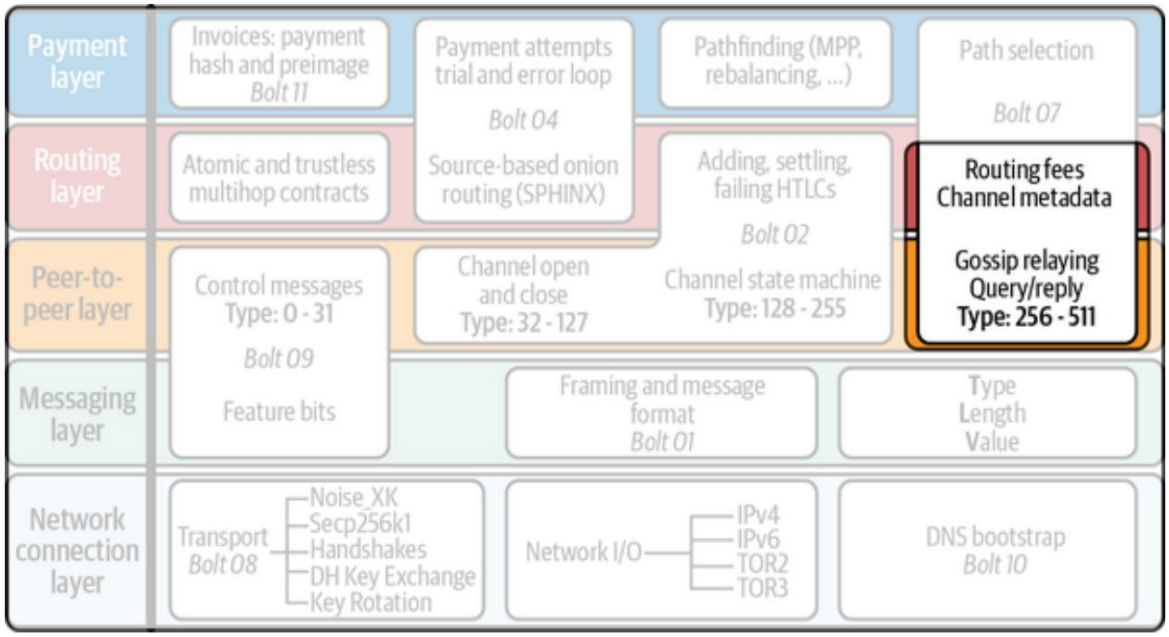

Figura 11-1. Protocolo Gossip en el conjunto de protocolos Lightning

Como ya aprendimos, Lightning Network utiliza un protocolo de enrutamiento de cebolla basado en la fuente para entregar un pago de un remitente al destinatario. Para hacer esto, el nodo emisor debe poder construir una ruta de canales de pago que lo conecte con el destinatario, como veremos en el Capítulo 12. Por lo tanto, el remitente debe poder mapear Lightning Network construyendo un canal gráfico. . El ***gráfico de canales*** es el conjunto interconectado de canales anunciados públicamente y los nodos que estos canales interconectan.

Dado que los canales están respaldados por una transacción de financiación que tiene lugar en la cadena, uno podría creer falsamente que los nodos Lightning podrían simplemente extraer los canales existentes de la cadena de bloques de Bitcoin. Sin embargo, esto sólo es posible hasta cierto punto. Las transacciones de financiación son direcciones Pay-to-Witness Script-Hash (P2WSH), y la naturaleza de la secuencia de comandos (un multisig 2 de 2) solo se revelará una vez que se gaste el resultado de la transacción de financiación.
Incluso si se conociera la naturaleza del script, es importante recordar que no todos los scripts multigrado 2 de 2 corresponden a canales de pago.

Hay aún más razones por las que mirar la cadena de bloques de Bitcoin podría no ser útil. Por ejemplo, en Lightning Network, las claves de Bitcoin que se utilizan para firmar son rotadas por los nodos para cada canal y actualización. Por lo tanto, incluso si pudiéramos detectar de manera confiable las transacciones de financiación en la cadena de bloques de Bitcoin, no sabríamos qué dos nodos en Lightning Network poseen ese canal en particular.

Lightning Network resuelve este problema implementando un ***protocolo de chismes.***
Los protocolos de chismes son típicos de las redes peer-to-peer (P2P) y permiten que los nodos compartan información con toda la red con solo unas pocas conexiones directas con los pares. Los nodos Lightning abren conexiones cifradas de igual a igual entre sí y comparten (chismes) información que han recibido de otros pares. Tan pronto como un nodo quiere compartir información, por ejemplo, sobre un canal recién creado, envía un mensaje a todos sus pares. Al recibir un mensaje, un nodo decide si el mensaje recibido era nuevo y, de ser así, reenvía la información a sus pares. De esta forma, si la red peer-to-peer está bien conectada, toda la información nueva que sea necesaria para el funcionamiento de la red eventualmente se propagará a todos los demás pares.

Obviamente, si un nuevo par se une a la red por primera vez, necesita conocer a otros pares en la red para poder conectarse con otros y participar en la red.

En este capítulo, exploraremos exactamente ***cómo los*** nodos Lightning se descubren entre sí, descubren y actualizan su estado de nodo y se comunican entre sí.

Cuando la mayoría se refiere a la parte de la ***red*** de Lightning Network, se refieren al ***gráfico de canales,*** que en sí mismo es una estructura de datos autenticada única ***anclada*** en la cadena de bloques base de Bitcoin.

Sin embargo, Lightning Network también es una red de nodos peer-to-peer que chismean información sobre canales y nodos de pago. Por lo general, para que dos pares mantengan un canal de pago, deben hablar entre ellos directamente, lo que significa que habrá una conexión de pares entre ellos. Esto sugiere que el gráfico de canales es una subred de la red peer-to-peer.

Sin embargo, esto no es cierto porque los canales de pago pueden permanecer abiertos incluso si uno o ambos pares se desconectan temporalmente.

Revisemos parte de la terminología que hemos usado a lo largo del libro, específicamente mirando lo que significan en términos del gráfico de canal y la red peer-to-peer (vea la Tabla 11-1).

r

y

norte

t

norte

y

t

en

o

r

k

s

Gráfico de canal Red punto a punto	
canal	conexión
abierto	conectar
cerca	desconectar
transacción de financiación	conexión TCP/IP cifrada
enviar	transmitir
pago	mensaje

Debido a que Lightning Network es una red de igual a igual, se requiere un arranque inicial para que los pares se descubran entre sí. En este capítulo, seguiremos la historia de un nuevo par que se conecta a la red por primera vez y examinaremos cada paso en el proceso de arranque, desde el descubrimiento inicial del par hasta la sincronización y validación del gráfico de canal.

Como paso inicial, nuestro nuevo nodo necesita ***descubrir*** de alguna manera al menos ***un*** par que ya esté conectado a la red y tenga un gráfico de canales completo (como veremos más adelante, no hay una versión canónica del gráfico de canales). Usando

uno de los muchos protocolos iniciales de arranque para encontrar ese primer par, después de que se establece una conexión, nuestro nuevo par ahora necesita ***descargar*** y ***validar*** el gráfico de canal. Una vez que el gráfico de canales se ha validado por completo, nuestro nuevo par está listo para comenzar a abrir canales y enviar pagos en la red.

Después del arranque inicial, un nodo en la red debe continuar manteniendo su vista del gráfico de canales mediante el procesamiento de nuevas actualizaciones de políticas de enrutamiento de canales, el descubrimiento y la validación de nuevos canales, la eliminación de canales que se han cerrado en la cadena y, finalmente, la eliminación de canales que fallan. para enviar un "latido del corazón" adecuado cada dos semanas más o menos.

Al finalizar este capítulo, comprenderá un componente clave de Lightning Network peer-to-peer: a saber, cómo los pares se descubren entre sí y mantienen una copia local (perspectiva) del gráfico de canal. Comenzaremos explorando la historia de un nuevo nodo que acaba de iniciarse y necesita encontrar otros pares para conectarse a la red.

Descubrimiento de pares

En esta sección, comenzaremos a seguir un nuevo nodo Lightning que desee unirse a la red a través de tres pasos:

1. Descubra un conjunto de pares de arranque
2. Descarga y valida el gráfico de canales
3. Comenzar el proceso de mantenimiento continuo del gráfico de canales
   sí mismo

Arranque P2P

Antes de hacer cualquier otra cosa, nuestro nuevo nodo primero necesita descubrir un conjunto de pares que ya forman parte de la red. Llamamos a este proceso arranque entre pares inicial, y es algo que cada red entre pares debe implementar correctamente para garantizar una red robusta y saludable.

Arrancar nuevos pares a las redes peer-to-peer existentes es un problema muy bien estudiado con varias soluciones conocidas, cada una con sus propias compensaciones distintas. La solución más simple a este problema es simplemente empaquetar un conjunto de pares de arranque ***codificados*** en el software de nodo P2P empaquetado. Esto es simple en el sentido de que cada nuevo nodo tiene una lista de pares de arranque en el software que están ejecutando, pero bastante frágil dado que si el conjunto de pares de arranque se desconecta, ningún nodo nuevo podrá unirse a la red. Debido a esta fragilidad, esta opción suele utilizarse como respaldo en caso de que ninguno de los otros mecanismos de arranque P2P funcione correctamente.

En lugar de codificar el conjunto de pares de arranque dentro del propio software/binario, podemos permitir que los pares obtengan dinámicamente un conjunto nuevo/nuevo de pares de arranque que pueden usar para unirse a la red. Llamaremos a este proceso ***descubrimiento inicial de pares.*** Por lo general, aprovecharemos los protocolos de Internet existentes para mantener y distribuir un conjunto de pares de arranque. Una lista no exhaustiva de protocolos que se han utilizado en el pasado para lograr el descubrimiento inicial de pares incluye:

- Servicio de nombres de dominio (DNS)
- Charla de retransmisión por Internet (IRC)
- Protocolo de transferencia de hipertexto (HTTP)

Similar al protocolo Bitcoin, el principal mecanismo inicial de descubrimiento de pares utilizado en Lightning Network ocurre a través de DNS. Debido a que el descubrimiento inicial de pares es una tarea crítica y universal para la red, el proceso se ha ***estandarizado*** en el BOLT #10: DNS Bootstrap.

Arranque de DNS

El PERNO #10 El documento describe una forma estandarizada de implementar el descubrimiento de pares utilizando el DNS. El tipo de arranque basado en DNS de Lightning utiliza hasta tres tipos de registros distintos:

- Registros SRV para descubrir un conjunto de ***claves públicas de nodo.***

- Un registro para asignar la clave pública de un nodo a su dirección IPv4 actual.
- Registros AAA para mapear la clave pública de un nodo a su dirección IPv6 actual.

Aquellos que estén algo familiarizados con el protocolo DNS pueden estar familiarizados con los tipos de registro A (nombre a dirección IPv4) y AAA (nombre a dirección IPv6), pero no con el tipo SRV. El tipo de registro SRV lo utilizan los protocolos creados sobre DNS para determinar la ***ubicación*** de un servicio específico. En nuestro contexto, el servicio en cuestión es un nodo Lightning determinado y la ubicación es su dirección IP. Necesitamos usar este tipo de registro adicional porque, a diferencia de los nodos dentro del protocolo Bitcoin, necesitamos tanto una clave pública ***como*** una dirección IP para conectarnos a un nodo. Como vemos en el Capítulo 13, el protocolo de cifrado de transporte utilizado en Lightning Network requiere el conocimiento de la clave pública de un nodo antes de conectarse, a fin de implementar la ocultación de identidad para los nodos en la red.

Flujo de trabajo de arranque de un nuevo compañero

Antes de profundizar en los detalles de BOLT #10, Primero describiremos el flujo de alto nivel de un nuevo nodo que desea usar el BOLT #10 para unirse a la red.

En primer lugar, un nodo debe identificar un solo servidor DNS o un conjunto de servidores DNS que entiendan el BOLT n.º 10 para que puedan usarse para el arranque P2P.

Si bien BOLT #10 usa ***lseed.bitcoinstats.com*** como servidor de semillas, no existe un conjunto "oficial" de semillas de DNS para este propósito, pero cada una de las implementaciones principales mantiene su propia semilla de DNS y consultan las semillas de los demás para fines de redundancia. En la Tabla 11-2 , verá una lista no exhaustiva de algunos servidores semilla DNS populares.

s
y
y
d
s
y
r
en
y
r
s

servidor DNS	mantenedor
lseed.bitcoinstats.com	cristian decker
nodos.lightning.directory	Laboratorios relámpago (Señor Osuntokun)
soa.nodes.lightning.directory	Laboratorios Lightning (Olaoluwa Osuntokun)
lseed.darosior.ninja	antoine punto

Las semillas de DNS existen tanto para la red principal como para la red de prueba de Bitcoin.

Por el bien de nuestro ejemplo, supondremos la existencia de una semilla de DNS BOLT #10 válida en ***nodes.lightning.directory.***

A continuación, nuestro nuevo nodo emitirá una consulta SRV para obtener un conjunto de ***compañeros de arranque candidatos.*** La respuesta a nuestra consulta será una serie de claves públicas codificadas en bech32. Debido a que DNS es un protocolo basado en texto, no podemos enviar datos binarios sin procesar, por lo que se requiere un esquema de codificación. BOLT #10 especifica una codificación bech32 debido a su uso en el ecosistema Bitcoin más amplio. El número de claves públicas codificadas devueltas depende del servidor que devuelve la consulta, así como de todos los resolutores que se interponen entre el cliente y el servidor autorizado.

Usando la herramienta de línea de comando de excavación ampliamente disponible, podemos consultar la versión de ***testnet*** de la semilla DNS mencionada anteriormente con el siguiente comando:

```
$ usted @8.8.8.8 test.nodes.lightning.directory SRV
```

Usamos el argumento @ para forzar la resolución a través del servidor de nombres de Google (con la dirección IP 8.8.8.8) porque no filtra las respuestas de consultas SRV grandes. Al final del comando, especificamos que solo queremos que se devuelvan los registros SRV. Una respuesta de muestra se parece al Ejemplo 11-1.

Ejemplo 11-1. Consultar la semilla DNS para nodos accesibles $ dig @8.8.8.8 test.nodes.lightning.directory SRV

```
; <<>> DiG 9.10.6 <<>> @8.8.8.8 test.nodes.lightning.directory SRV ; (1 servidor encontrado) ;; opciones globales: +cmd ;; Tengo
respuesta: ;; ->>HEADER<<- código de operación: CONSULTA, estado: NOERROR, id: 43610 ;; banderas: qr rd ra; CONSULTA:
1, RESPUESTA: 25, AUTORIDAD: 0, ADICIONAL: 1

;; SECCIÓN DE
PREGUNTAS: ;test.nodes.lightning.directory.                    EN          SRV

;; SECCIÓN DE RESPUESTAS:
test.nodes.lightning.directory. 59 EN SRV                                  10 10 9735 (1)
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test .nodes.lightning.directory. test.nodes.lightning.directory. 59
 EN SRV ln1qtgsl3efj8verd4z27k44xu0a59kncvsarxatafm334exgnuvwhnz8dkhx8.test .nodes.lightning.directory. (2)
                                                                           10 10 15735

   [...]

;; Tiempo de consulta: 89 ms ;; SERVIDOR:
8.8.8.8#53(8.8.8.8)
;; CUÁNDO: Jue 31 de diciembre 16:41:07 PST 2020
```

(1) Número de puerto TCP donde se puede alcanzar el nodo LN.

(2) Clave pública (ID) de nodo codificada como un nombre de dominio virtual.

Hemos truncado la respuesta por razones de brevedad y solo mostramos dos de las respuestas devueltas. Las respuestas contienen un nombre de dominio "virtual" para un nodo de destino, luego a la izquierda tenemos el ***puerto TCP*** donde se puede acceder a este nodo. La primera respuesta utiliza el puerto TCP estándar para Lightning Network: 9735.
La segunda respuesta usa un puerto personalizado, que está permitido por el protocolo.

A continuación, intentaremos obtener el otro dato que necesitamos para conectarnos a un nodo: su dirección IP. Sin embargo, antes de que podamos consultar esto, primero ***decodificaremos*** la codificación bech32 de la clave pública del nombre de dominio virtual:

```
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7
```

Decodificando esta cadena bech32 obtenemos la siguiente clave pública secp256k1 válida:

```
026c64f5a7f24c6f7f0e1d6ec877f23b2f672fb48967c2545f227d70636395eaf
3
```

Ahora que tenemos la clave pública sin procesar, le pediremos al servidor DNS que ***resuelva*** el host virtual proporcionado para que podamos obtener la información de IP (registro A) para el nodo, como se muestra en el Ejemplo 11-2.

Ejemplo 11-2. Obtención de la última dirección IP para un nodo

```
$ cavar
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test .nodes.lightning.directory A

; <<>> DiG 9.10.6 <<>>
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test .nodes.lightning.directory A ;; opciones globales:
 +cmd ;; Tengo respuesta:

;; ->>HEADER<<- código de operación: CONSULTA, estado: NOERROR, id: 41934 ;; banderas: qr rd ra;
CONSULTA: 1, RESPUESTA: 1, AUTORIDAD: 0, ADICIONAL: 1

;; PSEUDOSECCIÓN OPCIONAL: ;
EDNS: versión: 0, banderas:; upp:4096;; SECCIÓN DE

PREGUNTAS: ;ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.tes
```

```
t.nodes.lightning.directorio. EN UN

;; SECCIÓN DE RESPUESTAS:
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test .nodes.lightning.directory. 60 EN UN XXXX
                                                                                                          ❶

;; Tiempo de consulta: 83 ms
;; SERVIDOR: 2600:1700:6971:6dd0::1#53(2600:1700:6971:6dd0::1)
;; CUÁNDO: Jue 31 de diciembre 16:59:22 PST 2020
;; TAMAÑO MSG recibido: 138
```

❶ El servidor DNS devuelve una dirección IP *XXXX* . La reemplazamos con X en el texto aquí para evitar presentar una dirección IP real.

En el comando anterior, consultamos el servidor para poder obtener una dirección IPv4 (registro A) para nuestro nodo de destino (reemplazado por *XXXX* en el ejemplo anterior). Ahora que tenemos la clave pública sin procesar, la dirección IP y el puerto TCP, podemos conectarnos al protocolo de transporte del nodo en:

```
026c64f5a7f24c6f7f0e1d6ec877f23b2f672fb48967c2545f227d70636395eaf
3@XXXX:9735
```

Consultar el registro DNS A actual para un nodo determinado también se puede utilizar para buscar el ***último*** conjunto de direcciones. Estas consultas se pueden usar para sincronizar más rápidamente la información de direccionamiento más reciente para un nodo, en comparación con esperar actualizaciones de direcciones en la red de chismes (consulte “El mensaje de anuncio de nodo”).

En este punto de nuestro viaje, nuestro nuevo nodo Lightning encontró su primer par y estableció su primera conexión. Ahora podemos comenzar la segunda fase del nuevo arranque entre pares: sincronización y validación de gráficos de canales.

Primero, exploraremos más de las complejidades del propio BOLT #10 para analizar más a fondo cómo funcionan las cosas debajo del capó.

Opciones de consulta SRV

El PERNO #10 El estándar es altamente extensible debido a su uso de subdominios anidados como una capa de comunicación para opciones de consulta adicionales. los

El protocolo de arranque permite a los clientes especificar aún más el ***tipo*** de nodos que intentan consultar en comparación con el valor predeterminado de recibir un subconjunto aleatorio de nodos en las respuestas de consulta.

El esquema de subdominio de opción de consulta utiliza una serie de pares clave-valor donde la clave en sí es una ***sola letra*** y el conjunto de texto restante es el valor en sí. Los siguientes tipos de consulta existen en la versión actual del documento de estándares BOLT #10 :

r

El byte de ***reino*** que se utiliza para determinar qué cadenas o consultas de reino deben devolverse. Tal como está, el único valor para esta clave es 0, que denota "Bitcoin".

a

Permite a los clientes filtrar los nodos devueltos según los ***tipos*** de direcciones que anuncian. Como ejemplo, esto se puede usar para obtener solo nodos que anuncien una dirección IPv6 válida. El valor que sigue a este tipo se basa en un campo de bits que se ***indexa*** en el conjunto de ***tipos*** de direcciones especificados que se definen en BOLT #7. El valor predeterminado para este campo es 6, que representa tanto IPv4 como IPv6 (se establecen los bits 1 y 2).

yo

Una clave pública de nodo válida serializada en formato comprimido. Esto permite que un cliente consulte un nodo específico en lugar de recibir un conjunto de nodos aleatorios.

norte

El número de registros a devolver. El valor predeterminado para este campo es 25.

Una consulta de ejemplo con opciones de consulta adicionales se parece a lo siguiente:

```
r0.a2.n10.nodes.lightning.directorio
```

Desglosando la consulta un par clave-valor a la vez, obtenemos los siguientes conocimientos:

r0

La consulta apunta al reino de Bitcoin

a2

La consulta solo quiere que se devuelvan direcciones IPv4

n10

Las solicitudes de consulta

Pruebe usted mismo algunas combinaciones de las distintas banderas utilizando la herramienta de línea de comandos dig DNS:

usted @ 8.8.8.8 r0.a6.nodes.lightning.directory SRV

El gráfico de canal

Ahora que nuestro nuevo nodo puede usar el protocolo de arranque de DNS para conectarse a su primer par, ¡puede comenzar a sincronizar el gráfico del canal! Sin embargo, antes de sincronizar el gráfico de canales, necesitaremos saber exactamente a ***qué*** nos referimos con gráfico de canales. En esta sección, exploraremos la ***estructura*** precisa del gráfico de canal y examinaremos los aspectos únicos del gráfico de canal en comparación con la estructura de datos de "gráfico" abstracto típico que es bien conocido/utilizado en el campo de la informática.

Un gráfico dirigido

Un ***gráfico*** en informática es una estructura de datos especial compuesta de vértices (normalmente denominados nodos) y bordes (también conocidos como enlaces). Dos nodos pueden estar conectados por uno o más bordes. El gráfico de canal también está ***dirigido*** dado que un pago puede fluir en cualquier dirección a lo largo de un

borde dado (un canal). En la figura 11-2 se muestra un ejemplo de un ***gráfico dirigido .***

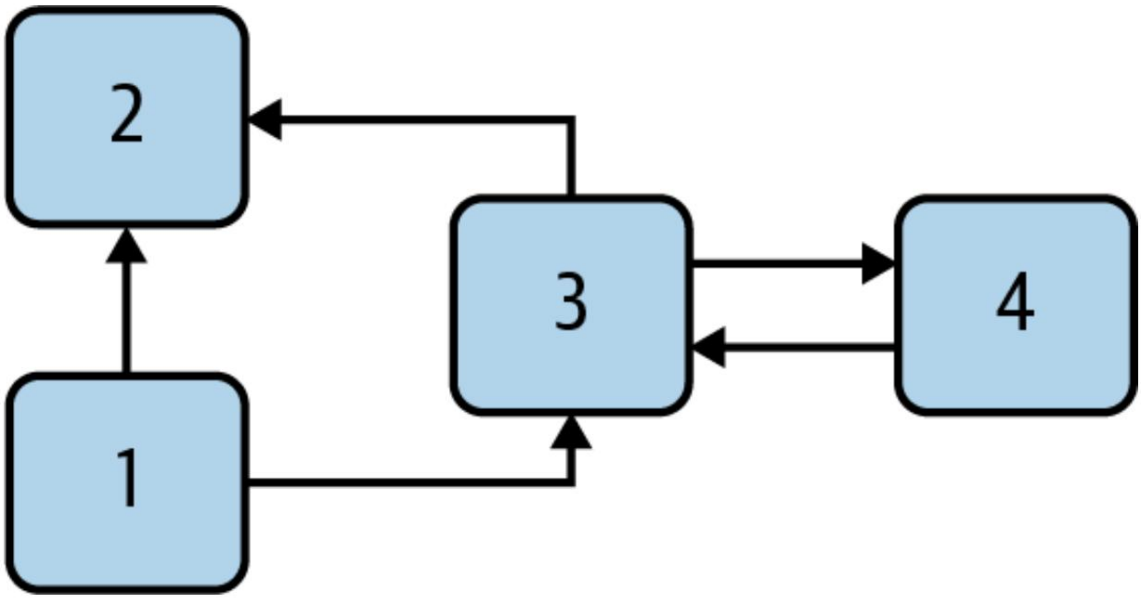

Figura 11-2. Un gráfico dirigido

En el contexto de Lightning Network, nuestros vértices son los propios nodos Lightning, y nuestros bordes son los canales de pago que conectan estos nodos. Debido a que nos preocupa el ***enrutamiento de los pagos,*** en nuestro modelo, un nodo sin bordes (sin canales de pago) no se considera parte del gráfico, ya que no es útil.

Debido a que los propios canales son UTXO (direcciones multigrado financiadas 2 de 2), podemos ver el gráfico de canal como un subconjunto especial del conjunto Bitcoin UTXO, además del cual podemos agregar información adicional (los nodos, etc.) para llegar a la estructura superpuesta final, que es el gráfico de canal. Este anclaje de los componentes fundamentales del gráfico de canal en la cadena de bloques base de Bitcoin significa que es imposible ***falsificar*** un gráfico de canal válido, que tiene propiedades útiles en lo que respecta a la prevención de spam, como veremos más adelante.

Mensajes de protocolo de chismes

La información del gráfico de canal se propaga a través de Lightning P2P Red como tres mensajes, que se describen en el TORNILLO #7:

anuncio_nodo

> El vértice en nuestro gráfico que comunica la clave pública de un nodo, así como también cómo llegar al nodo a través de Internet y algunos metadatos adicionales que describen el conjunto de ***características*** que admite el nodo.

canal_anuncio

> Una prueba anclada en blockchain de la existencia de un canal entre dos nodos individuales. Cualquier tercero puede verificar esta prueba para asegurarse de que realmente se anuncia un canal ***real .*** Similar a node_announcement, este mensaje también contiene información que describe las ***capacidades*** del canal, lo cual es útil cuando se intenta enrutar un pago.

canal_actualizar

> Un ***par*** de estructuras que describen el conjunto de políticas de enrutamiento para un canal dado. Los mensajes channel_update vienen en ***pares*** porque un canal es un borde dirigido, por lo que cada lado del canal puede especificar su propia política de enrutamiento personalizada.

Es importante tener en cuenta que cada componente del gráfico de canal está ***autenticado,*** lo que permite que un tercero se asegure de que el propietario de un canal/actualización/nodo es realmente el que envía una actualización. Esto hace que el gráfico de canal sea un tipo único de ***estructura de datos autenticado*** que no se puede falsificar. Para la autenticación, usamos una firma digital ECDSA secp256k1 (o una serie de ellas) sobre el resumen serializado del mensaje en sí. No entraremos en detalles específicos del encuadre/serialización de mensajes utilizados en Lightning Network en este capítulo, ya que cubriremos esa información en el Capítulo 13.

Con la estructura de alto nivel del gráfico de canales presentada, ahora nos sumergiremos en la estructura precisa de cada uno de los tres mensajes que se usan para chismear.

el gráfico del canal. También explicaremos cómo se puede verificar cada mensaje y componente del gráfico de canal.

El mensaje de anuncio de nodo

Primero, tenemos el mensaje node_announcement, que tiene dos propósitos principales:

1. Para anunciar información de conexión para que otros nodos puedan conectarse un nodo para arrancar en la red o para intentar establecer un nuevo canal de pago con ese nodo.
2. Comunicar el conjunto de funciones (capacidades) a nivel de protocolo que un nodo comprende/admite. La negociación de funciones entre nodos permite a los desarrolladores agregar nuevas funciones de forma independiente y admitirlas con cualquier otro nodo de forma opcional.

A diferencia de los anuncios de canales, los anuncios de nodos no están anclados en la cadena de bloques base. Por lo tanto, los anuncios de nodo solo se consideran válidos si se han propagado con un anuncio de canal correspondiente.
En otras palabras, siempre rechazamos los nodos sin canales de pago para garantizar que un par malintencionado no pueda inundar la red con nodos falsos que no forman parte del gráfico de canales.

La estructura del mensaje node_announcement

El node_announcement se compone de los siguientes campos:

firma

Una firma ECDSA válida que cubra el resumen serializado de todos los campos enumerados a continuación. Esta firma debe corresponder a la clave pública del nodo anunciado.

caracteristicas

Un vector de bits que describe el conjunto de funciones de protocolo que comprende este nodo. Cubriremos este campo con más detalle en "Bits de características y

Extensibilidad del protocolo” sobre la extensibilidad del protocolo Lightning. En un nivel alto, este campo lleva un conjunto de bits que representan las funciones que comprende un nodo. Como ejemplo, un nodo puede indicar que comprende el último tipo de canal.

marca de tiempo

Una marca de tiempo codificada de época Unix. Esto permite a los clientes imponer un ordenamiento parcial sobre las actualizaciones del anuncio de un nodo.

nodo_id

La clave pública secp256k1 a la que pertenece este anuncio de nodo.
Solo puede haber un único anuncio de nodo para un nodo determinado en el gráfico de canal en un momento dado. Como resultado, un anuncio de nodo puede reemplazar un anuncio de nodo anterior para el mismo nodo si lleva una marca de tiempo más alta (posterior).

color_rgb

Un campo que permite que un nodo especifique un color RGB para asociarlo, a menudo utilizado en visualizaciones de gráficos de canales y directorios de nodos.

alias

Una cadena UTF-8 que sirve como apodo para un nodo determinado. Tenga en cuenta que no se requiere que estos alias sean únicos globalmente, ni se verifican de ninguna manera. Como resultado, no se debe confiar en ellos como una forma de identidad; pueden falsificarse fácilmente.

direcciones

Un conjunto de direcciones accesibles de Internet públicas que se asociarán con un nodo determinado. En la versión actual del protocolo, se admiten cuatro tipos de direcciones: IPv4 (tipo: 1), IPv6 (tipo: 2), Tor v2 (tipo: 3) y Tor v3 (tipo: 4). En el mensaje de anuncio de nodo, cada uno de estos

tipos de dirección se denota por un tipo de número entero que se incluye entre paréntesis después del tipo de dirección.

Validación de anuncios de nodos

La validación de un anuncio de nodo entrante es sencilla. Las siguientes afirmaciones deben confirmarse al examinar un anuncio de nodo:

- Si ya se conoce un nodo_anuncio existente para ese nodo, entonces el campo de marca de tiempo de un nuevo nodo_anuncio entrante debe ser mayor que el anterior.
- Con esta restricción, imponemos un nivel forzado de "frescura".
- Si no existe un anuncio de nodo para el nodo dado, entonces un anuncio de canal existente que hace referencia al nodo dado (más sobre eso más adelante) ya debe existir en el gráfico de canal local de uno.
- La firma incluida debe ser una firma ECDSA válida verificada mediante la clave pública node_id incluida y el resumen SHA-256 doble de la codificación del mensaje sin procesar (menos la firma y el encabezado del marco) como mensaje.
- Todas las direcciones incluidas deben ordenarse en orden ascendente según su identificador de dirección.
- Los bytes de alias incluidos deben ser una cadena UTF-8 válida.

El mensaje channel_announcement

A continuación, tenemos el mensaje channel_announcement, que se utiliza para ***anunciar*** un nuevo canal ***público*** a la red más amplia. Tenga en cuenta que anunciar un canal es ***opcional.*** Solo es necesario anunciar un canal si está destinado a ser utilizado para el enrutamiento por Lightning Network. Es posible que los nodos de enrutamiento activos deseen anunciar todos sus canales. Sin embargo, ciertos nodos como mobile

es probable que los nodos no tengan el tiempo de actividad o el deseo de ser un nodo de enrutamiento activo. Como resultado, estos nodos móviles (que generalmente usan clientes ligeros para conectarse a la red P2P de Bitcoin) pueden tener canales (privados) puramente ***no anunciados .***

Canales no anunciados (privados)

Un canal no anunciado no forma parte del gráfico de canales públicos conocidos, pero aún puede usarse para enviar o recibir pagos. Un lector astuto ahora puede preguntarse cómo un canal que no forma parte del gráfico de canales públicos puede recibir pagos. La solución a este problema es un conjunto de "ayudantes de búsqueda de ruta" que llamamos sugerencias de enrutamiento. Como veremos en el Capítulo 15, las facturas creadas por nodos con canales no anunciados incluirán información para ayudar al remitente a enrutarlas, asumiendo que el nodo tiene al menos un único canal con un nodo de enrutamiento público existente.

Debido a la existencia de canales no anunciados, se desconoce el tamaño ***real*** del gráfico de canales (tanto el componente público como el privado).

Localización de un canal en la cadena de bloques de bitcoin

Como se mencionó anteriormente, el gráfico de canales se autentica debido a su uso de criptografía de clave pública, así como a la cadena de bloques de Bitcoin como sistema de prevención de spam. Para que un nodo acepte un nuevo anuncio de canal, el anuncio debe ***demostrar*** que el canal realmente existe en la cadena de bloques de Bitcoin. Este sistema de prueba agrega un costo inicial para agregar una nueva entrada al gráfico del canal (las tarifas en cadena que se deben pagar para crear el UTXO del canal). Como resultado, mitigamos el spam y nos aseguramos de que un nodo deshonesto en la red no pueda llenar la memoria de un nodo honesto sin costo alguno con canales falsos.

Dado que necesitamos construir una prueba de la existencia de un canal, una pregunta natural que surge es: ¿cómo "señalamos" o hacemos referencia a un canal determinado para el verificador? Dado que un canal de pago está anclado en una salida de transacción no gastada (consulte "Entradas y salidas"), un pensamiento inicial podría ser intentar primero anunciar el punto de salida completo (txid: índice) del canal. Dado que el outpoint es globalmente único y confirmado en el

cadena, esto suena como una buena idea; sin embargo, tiene un inconveniente: el verificador debe mantener una copia completa del conjunto de UTXO para verificar los canales. Esto funciona bien para los nodos completos de Bitcoin, pero los clientes que dependen de una verificación ligera no suelen mantener un conjunto completo de UTXO. Debido a que queremos asegurarnos de que podemos admitir nodos móviles en Lightning Network, nos vemos obligados a encontrar otra solución.

¿Qué pasa si en lugar de hacer referencia a un canal por su UTXO, lo hacemos en función de su "ubicación" en la cadena? Para hacer esto, necesitaremos un esquema que nos permita hacer referencia a un bloque determinado, luego a una transacción dentro de ese bloque y finalmente a una salida específica creada por esa transacción. Tal identificador se describe en el TORNILLO #7 y se conoce como ***ID de canal corto*** o scid.

El scid se usa en channel_announcement (y channel_update), así como dentro del paquete de enrutamiento con cifrado de cebolla incluido dentro de los HTLC, como aprendimos en el Capítulo 10.

El ID de canal corto

Con base en la información anterior, tenemos tres piezas de información que necesitamos codificar para hacer referencia de manera única a un canal determinado. Como queremos una representación compacta, intentaremos codificar la información en un ***solo*** número entero. Nuestro formato de número entero de elección es un número entero de 64 bits sin signo, compuesto por 8 bytes.

Primero, la altura del bloque. Utilizando 3 bytes (24 bits) podemos codificar 16.777.216 bloques. Eso nos deja 5 bytes para codificar el índice de transacción y el índice de salida, respectivamente. Usaremos los siguientes 3 bytes para codificar el índice de transacción ***dentro*** de un bloque. Esto es más que suficiente dado que solo es posible corregir decenas de miles de transacciones en un bloque con los tamaños de bloque actuales. Esto nos deja 2 bytes para codificar el índice de salida del canal dentro de la transacción.

Nuestro formato scid final se parece a:

bloque_altura (3 bytes) || índice_transacción (3 bytes) || índice_salida (2 bytes)

Usando técnicas de empaquetamiento de bits, primero codificamos los 3 bytes más significativos como la altura del bloque, los siguientes 3 bytes como el índice de transacción y los 2 bytes menos significativos como el índice de salida que crea el canal UTXO.

Un ID de canal corto se puede representar como un solo número entero (695313561322258433) o como una cadena más amigable para los humanos: 632384x1568x1. Aquí vemos que el canal se extrajo en el bloque 632384, fue la transacción 1568 del bloque, con la salida del canal como la segunda salida (los UTXO están indexados a cero) producida por la transacción.

Ahora que podemos señalar sucintamente una salida de financiación de canal determinada en la cadena, podemos examinar la estructura completa del mensaje channel_announcement, así como ver cómo verificar la prueba de existencia incluida en el mensaje.

La estructura del mensaje channel_announcement

Un channel_announcement comunica principalmente dos cosas:

1. Una prueba de que existe un canal entre el nodo A y el nodo B con ambos nodos controlando las teclas multisig en la salida de ese canal.
2. El conjunto de capacidades del canal (qué tipos de HTLC puede ruta, etc).

Al describir la prueba, generalmente nos referiremos al nodo 1 y al nodo 2. De los dos nodos que conecta un canal, el "primer" nodo es el nodo que tiene una codificación de clave pública "inferior" cuando comparamos la clave pública de los dos nodos en formato comprimido con codificación hexadecimal en orden lexicográfico.
En consecuencia, además de una clave pública de nodo en la red, cada nodo también debe controlar una clave pública dentro de la cadena de bloques de Bitcoin.

De manera similar al mensaje node_announcement, todas las firmas incluidas del mensaje channel_announcement deben firmarse/verificarse con la codificación sin procesar del mensaje (menos el encabezado) que sigue ***a*** la firma final (porque no es posible que una firma digital se firme por sí misma).).

Dicho esto, un mensaje channel_announcement tiene los siguientes campos:

firma_nodo_1

La firma del primer nodo sobre el resumen del mensaje.

firma_nodo_2

La firma del segundo nodo sobre el resumen del mensaje.

bitcoin_signature_1

La firma de la clave multisig (en la salida de financiación) del primer nodo sobre el resumen del mensaje.

firma_bitcoin_2

La firma de la clave multisig (en la salida de financiación) del segundo nodo sobre el resumen del mensaje.

caracteristicas

Un vector de bits de característica que describe el conjunto de características de nivel de protocolo admitidas por este canal.

cadena_hash

Un hash de 32 bytes que suele ser el hash del bloque de génesis de la cadena de bloques (por ejemplo, la red principal de Bitcoin) en el que se abrió el canal.

short_channel_id

El scid que ubica de manera única la salida de financiamiento del canal dado dentro de la cadena de bloques.

nodo_id_1

La clave pública del primer nodo de la red.

nodo_id_2

La clave pública del segundo nodo de la red.

bitcoin_key_1

La clave multisig sin procesar para la salida de financiación del canal para el primer nodo de la red.

bitcoin_key_2

La clave multisig sin procesar para la salida de financiación del canal para el segundo nodo de la red.

Validación de anuncio de canal

Ahora que sabemos qué contiene un anuncio de canal, podemos ver cómo verificar la existencia del canal en la cadena.

Armado con la información en channel_announcement, cualquier nodo Lightning (incluso uno sin una copia completa de la cadena de bloques de Bitcoin) puede verificar la existencia y autenticidad del canal de pago.

Primero, el verificador usará la ID de canal corta para encontrar qué bloque de Bitcoin contiene la salida de financiación del canal. Con la información de la altura del bloque, el verificador puede solicitar solo ese bloque específico de un nodo de Bitcoin. Luego, el bloque se puede volver a vincular al bloque de génesis siguiendo la cadena del encabezado del bloque hacia atrás (verificando la prueba de trabajo), confirmando que este es, de hecho, un bloque que pertenece a la cadena de bloques de Bitcoin.

A continuación, el verificador utiliza el número de índice de la transacción para identificar el ID de transacción de la transacción que contiene el canal de pago. La mayoría de las bibliotecas modernas de Bitcoin permitirán la indexación en la transacción de un bloque en función del índice de la transacción dentro del bloque mayor.

A continuación, el verificador utiliza una biblioteca de Bitcoin (en el idioma del verificador) para extraer la transacción relevante según su índice dentro del bloque. El verificador validará la transacción (comprobando que esté debidamente firmada y produzca el mismo ID de transacción cuando se le aplica un hash).

A continuación, el verificador extraerá la salida Pay-to-Witness-Script-Hash (P2WSH) a la que hace referencia el número de índice de salida del ID de canal corto. Esta es la dirección de la salida de financiación del canal. Además, el verificador se asegurará de que el tamaño del presunto canal coincida con el valor de la salida producida en el índice de salida especificado.

Finalmente, el verificador reconstruirá el script multisig de bitcoin_key_1 y bitcoin_key_2 y confirmará que produce la misma dirección que en la salida.

¡El verificador ahora ha verificado de forma independiente que el canal de pago en el anuncio está financiado y confirmado en la cadena de bloques de Bitcoin!

El mensaje channel_update

El tercer y último mensaje utilizado en el protocolo de chismes es el mensaje channel_update. Se generan dos de estos para cada canal de pago (uno por cada socio de canal) que anuncian sus tarifas de enrutamiento, expectativas de bloqueo de tiempo y capacidades.

El mensaje channel_update también contiene una marca de tiempo, lo que permite que un nodo actualice sus tarifas de enrutamiento y otras expectativas y capacidades mediante el envío de un nuevo mensaje channel_update con una marca de tiempo más alta (posterior) que reemplaza cualquier actualización anterior.

El mensaje channel_update contiene los siguientes campos:

firma

Una firma digital que coincida con la clave pública del nodo, para autenticar la fuente y la integridad de la actualización del canal.

cadena_hash

El hash del bloque de génesis de la cadena que contiene el canal.

short_channel_id

El ID de canal corto para identificar el canal.

marca de tiempo

La marca de tiempo de esta actualización, para permitir a los destinatarios secuenciar actualizaciones y reemplazar actualizaciones anteriores

mensaje_banderas

Un campo de bits que indica la presencia de campos adicionales en el mensaje channel_update

banderas_del_canal

Un campo de bits que muestra la dirección del canal y otras opciones de canales

cltv_expiry_delta

Las expectativas delta de bloqueo de tiempo de este nodo para el enrutamiento (ver Capítulo 10)

htlc_minimum_msat

La cantidad mínima de HTLC que se enrutará

fee_base_msat

La tarifa base que se cobrará por el enrutamiento

tarifa_proporcional_millonésimas

La tasa de tarifa proporcional que se cobrará por el enrutamiento

htlc_maximum_msat (opción_canal_htlc_max)

La cantidad máxima que se enrutará

Un nodo que recibe el mensaje channel_update puede adjuntar estos metadatos al borde del gráfico de canal para permitir la búsqueda de rutas, como veremos en el Capítulo 12.

Mantenimiento continuo de gráficos de canales

La construcción de un gráfico de canal no es un evento de una sola vez, sino una actividad continua. A medida que un nodo se inicia en la red, comenzará a recibir "chismes", en forma de tres mensajes de actualización. Utilizará estos mensajes para comenzar inmediatamente a construir un gráfico de canal validado.

Cuanta más información recibe un nodo, mejor se vuelve su "mapa" de Lightning Network y más efectivo puede ser en la búsqueda de rutas y la entrega de pagos.

Un nodo no solo agregará información al gráfico del canal. También hará un seguimiento de la última vez que se actualizó un canal y eliminará los canales "obsoletos" que no se hayan actualizado en más de dos semanas. Finalmente, si ve que algún nodo ya no tiene canales, también eliminará ese nodo.

La información recopilada del protocolo de chismes no es la única información que se puede almacenar en el gráfico de canales. Diferentes implementaciones de nodos Lightning pueden adjuntar otros metadatos a nodos y canales.
Por ejemplo, algunas implementaciones de nodos calculan una "puntuación" que evalúa la "calidad" de un nodo como par de enrutamiento. Esta puntuación se utiliza como parte de la búsqueda de rutas para priorizar o quitar prioridad a las rutas.

Conclusión

En este capítulo, aprendimos cómo los nodos Lightning se detectan entre sí, descubren y actualizan su estado de nodo y se comunican entre sí.
Hemos aprendido cómo se crean y mantienen los gráficos de canales, y hemos explorado algunas formas en que Lightning Network disuade a los malos actores o nodos deshonestos de enviar spam a la red.

Capítulo 12. Búsqueda de caminos y entrega de pagos

La entrega de pagos en Lightning Network depende de encontrar una ruta desde el remitente hasta el destinatario, un proceso llamado búsqueda de ruta . Dado que el enrutamiento lo realiza el remitente, el remitente debe encontrar una ruta adecuada para llegar al destino. Esta ruta luego se codifica en una cebolla, como vimos en el Capítulo 10.

En este capítulo examinaremos el problema de la búsqueda de rutas, comprenderemos cómo la incertidumbre sobre los balances de canales complica este problema y veremos cómo una implementación típica de búsqueda de rutas intenta resolverlo.

Pathfinding en Lightning Protocol Suite

La búsqueda de rutas, la selección de rutas, los pagos de varias partes (MPP) y el bucle de prueba y error de intento de pago ocupan la mayor parte de la capa de pago en la parte superior del conjunto de protocolos.

Estos componentes se destacan mediante un esquema en el conjunto de protocolos, que se muestra en la Figura 12-1.

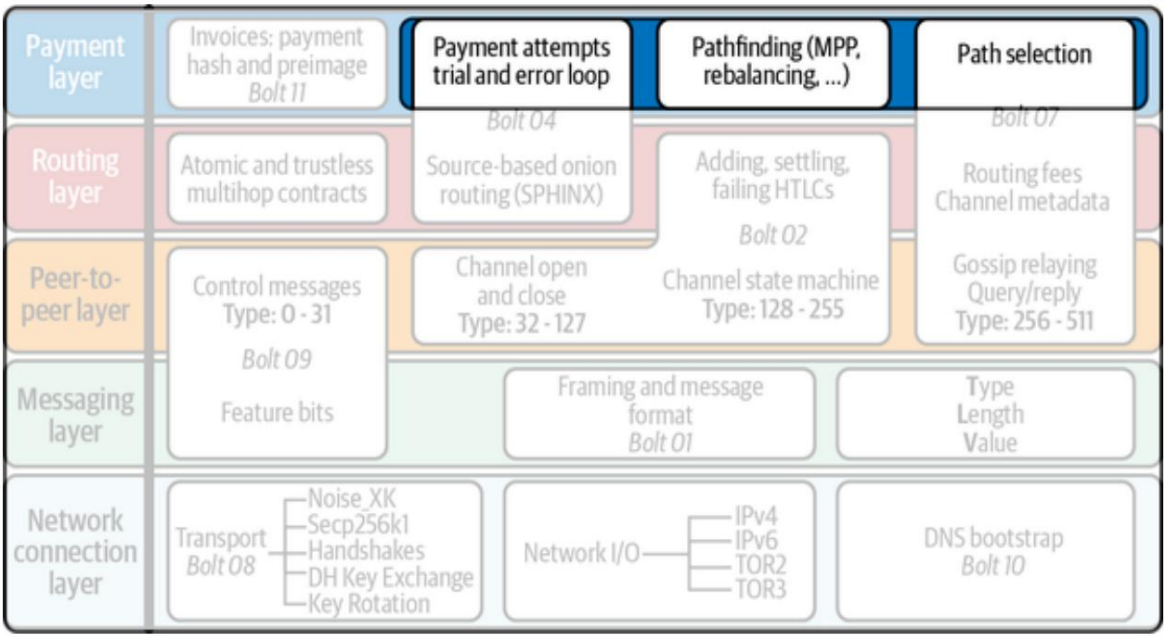

Figura 12-1. Entrega de pagos en el conjunto de protocolos Lightning

¿Dónde está el PERNO?

Hasta ahora, hemos analizado varias tecnologías que forman parte de Lightning Network y hemos visto su especificación exacta como parte de un estándar BOLT. ¡Se sorprenderá al descubrir que la búsqueda de caminos no es parte de los BOLT!

Esto se debe a que la búsqueda de rutas no es una actividad que requiera ningún tipo de coordinación o interoperabilidad entre diferentes implementaciones. Como hemos visto, la ruta es seleccionada por el remitente. Aunque los detalles de enrutamiento se especifican en detalle en los BOLT, el descubrimiento y la selección de la ruta se dejan completamente en manos del remitente. Entonces, cada implementación de nodo puede elegir una estrategia/algoritmo diferente para encontrar rutas. De hecho, las diferentes implementaciones de nodo/cliente y monedero pueden incluso competir y utilizar su algoritmo de búsqueda de rutas como punto de diferenciación.

Pathfinding: ¿Qué problema estamos resolviendo?

El término búsqueda de caminos puede ser algo engañoso porque implica la búsqueda de ***un solo camino que*** conecte dos nodos. Al principio, cuando el

Lightning Network era pequeña y no estaba bien interconectada, el problema era encontrar una manera de unirse a los canales de pago para llegar al destinatario.

Pero, a medida que Lightning Network ha crecido explosivamente, la naturaleza del problema de búsqueda de caminos ha cambiado. A mediados de 2021, cuando terminamos este libro, Lightning Network consta de 20 000 nodos conectados por al menos 55 000 canales públicos con una capacidad agregada de casi 2000 BTC. Un nodo tiene en promedio 8,8 canales, mientras que los 10 nodos más conectados tienen entre 400 y 2000 canales ***cada uno.*** En la Figura 12-2 se muestra una visualización de solo un pequeño subconjunto del gráfico de canal LN .

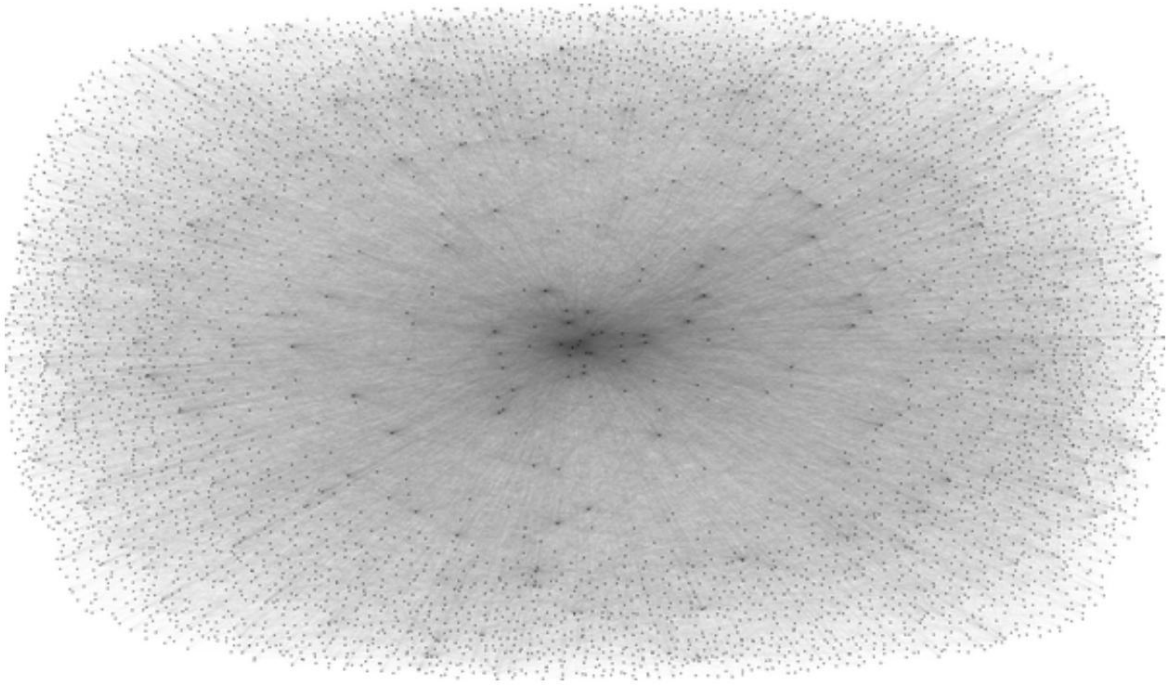

Figura 12-2. Una visualización de parte de Lightning Network a partir de julio de 2021

> **NOTA**
>
> La visualización de la red en la Figura 12-2 se produjo con un simple script de Python que puede encontrar en code/lngraph en el repositorio del libro.

Si el remitente y el destinatario están conectados a otros nodos bien conectados y tienen al menos un canal con la capacidad adecuada, habrá miles de

caminos. El problema se convierte en seleccionar la ***mejor*** ruta que tendrá éxito en la entrega del pago, de una lista de miles de rutas posibles.

Selección de la mejor ruta

Para seleccionar el mejor camino, primero tenemos que definir lo que queremos decir con "mejor". Puede haber muchos criterios diferentes, tales como:

- Caminos con suficiente liquidez. Obviamente, si una ruta no tiene suficiente liquidez para enrutar nuestro pago, entonces no es una ruta adecuada.
- Caminos con tarifas bajas. Si tenemos varios candidatos, es posible que queramos seleccionar aquellos con tarifas más bajas.
- Caminos con bloqueos de tiempo cortos. Es posible que deseemos evitar bloquear nuestros fondos durante demasiado tiempo y, por lo tanto, seleccionar rutas con bloqueos de tiempo más cortos.

Todos estos criterios pueden ser deseables hasta cierto punto, y seleccionar caminos que sean favorables en muchas dimensiones no es una tarea fácil. Los problemas de optimización como este pueden ser demasiado complejos para resolver la "mejor" solución, pero a menudo se pueden resolver con alguna aproximación de la óptima, lo cual es una buena noticia porque, de lo contrario, la búsqueda de caminos sería un problema intratable.

Pathfinding en Matemáticas y Ciencias de la Computación

Pathfinding in the Lightning Network cae dentro de una categoría general de teoría de ***grafos*** en matemáticas y la categoría más específica de ***recorrido de grafos*** en informática.

Una red como Lightning Network se puede representar como una construcción matemática llamada ***gráfico,*** donde los ***nodos*** están conectados entre sí por ***bordes*** (equivalentes a los canales de pago). Lightning Network forma un ***gráfico dirigido*** porque los nodos están vinculados de forma ***asimétrica,*** ya que el saldo del canal se divide entre los dos socios del canal y la liquidez de pago es diferente en cada dirección. Un grafo dirigido con restricciones de capacidad numérica en sus bordes se llama ***flujo .***

red, una construcción matemática utilizada para optimizar el transporte y otras redes similares. Las redes de flujo se pueden usar como marco cuando las soluciones necesitan lograr un flujo específico y minimizar el costo, conocido como el problema de flujo de costo mínimo (MCFP).

Capacidad, Saldo, Liquidez

Para comprender mejor el problema de transportar satoshis del punto A al punto B, debemos definir mejor tres términos importantes: capacidad, equilibrio y liquidez. Usamos estos términos para describir la capacidad de un canal de pago para enrutar un pago.

En un canal de pago que conecta AÿÿB:

Capacidad

> Esta es la cantidad total de satoshis que se financiaron en el multigrado 2 de 2 con la transacción de financiación. Representa la cantidad máxima de valor retenida en el canal. La capacidad del canal es anunciada por el protocolo de chismes y es conocida por los nodos.

Balance

> Esta es la cantidad de satoshis que tiene cada socio de canal que se puede enviar al otro socio de canal. Un subconjunto del saldo de A se puede enviar en la dirección (AÿB) hacia el nodo B. Un subconjunto del saldo de B se puede enviar en la dirección opuesta (AÿB).

Liquidez

> El saldo disponible (subconjunto) que realmente se puede enviar a través del canal en una dirección. La liquidez de A es igual al saldo de A menos la reserva del canal y cualquier HTLC pendiente comprometido por A.

El único valor conocido por la red (a través de chismes) es la capacidad total del canal. Una parte desconocida de esa capacidad se distribuye como el saldo de cada socio. Algún subconjunto de ese saldo está disponible para enviar a través del canal en una dirección:

capacidad = saldo(A) + saldo(B)

liquidez (A) = saldo (A) - reserva_canal (A) - HTLC pendientes (A)

Incertidumbre de Saldos

Si supiéramos los saldos exactos de los canales de cada canal, podríamos calcular una o más rutas de pago utilizando cualquiera de los algoritmos de búsqueda de rutas estándar que se enseñan en los buenos programas de informática. Pero no conocemos los saldos de los canales; solo conocemos la capacidad agregada del canal, que anuncian los nodos en los anuncios del canal. Para que un pago tenga éxito, debe haber un saldo adecuado en el lado de envío del canal.
Si no sabemos cómo se distribuye la capacidad entre los socios de canal, no sabemos si hay saldo suficiente en la dirección en la que intentamos enviar el pago.

Los saldos no se anuncian en las actualizaciones del canal por dos razones: privacidad y escalabilidad. Primero, anunciar los saldos reduciría la privacidad de Lightning Network porque permitiría la vigilancia del pago mediante el análisis estadístico de los cambios en los saldos. En segundo lugar, si los nodos anunciaran saldos (globalmente) con cada pago, la escala de Lightning Network sería tan mala como la de las transacciones de Bitcoin en cadena que se transmiten a todos los participantes. Por lo tanto, los saldos no se anuncian. Para resolver el problema de la búsqueda de caminos frente a la incertidumbre de los saldos, necesitamos estrategias innovadoras de búsqueda de caminos. Estas estrategias deben relacionarse estrechamente con el algoritmo de enrutamiento que se utiliza, que es el enrutamiento de cebolla basado en la fuente donde es responsabilidad del remitente encontrar una ruta a través de la red.

El problema de la incertidumbre se puede describir matemáticamente como un ***rango de liquidez,*** que indica los límites inferior y superior de liquidez en función de la información que se conoce. Dado que conocemos la capacidad del canal y conocemos el saldo de reserva del canal (el saldo mínimo permitido en cada extremo), la liquidez se puede definir como:

min(liquidez) = channel_reserve

max(liquidez) = capacidad – channel_reserve

o como un rango:

channel_reserve <= liquidez <= (capacidad – channel_reserve)

Nuestro rango de incertidumbre de liquidez del canal es el rango entre la liquidez mínima y máxima posible. Esto es desconocido para la red, excepto los dos socios de canal. Sin embargo, como veremos, podemos usar los HTLC fallidos devueltos de nuestros intentos de pago para actualizar nuestra estimación de liquidez y reducir la incertidumbre. Si, por ejemplo, recibimos un código de falla de HTLC que nos dice que un canal no puede cumplir con un HTLC que es más pequeño que nuestra estimación de liquidez máxima, eso significa que la liquidez máxima puede actualizarse al monto del HTLC fallido. En términos más simples, si creemos que la liquidez puede manejar un HTLC de ***N*** satoshis y descubrimos que no entrega ***M*** satoshis (donde ***M*** es menor), entonces podemos actualizar nuestra estimación a M–1 como límite superior. Intentamos encontrar el techo y chocamos contra él, ¡así que es más bajo de lo que pensábamos!

Complejidad pionera

Encontrar una ruta a través de un gráfico es un problema que las computadoras modernas pueden resolver de manera bastante eficiente. Los desarrolladores eligen principalmente la búsqueda en amplitud si todos los bordes tienen el mismo peso. En los casos en que los bordes no tienen el mismo peso, se utiliza un algoritmo basado en el algoritmo de Dijkstra, como A* (pronunciado “A-estrella”). En nuestro caso, los pesos de los bordes pueden representar las tarifas de enrutamiento. Solo se incluirán en la búsqueda los bordes con una capacidad superior a la cantidad a enviar. En esta forma básica, la búsqueda de rutas en Lightning Network es muy simple y directa.

Sin embargo, la liquidez del canal es desconocida para el remitente. Esto convierte nuestro sencillo problema teórico de informática en un problema bastante complejo del mundo real. Ahora tenemos que resolver un problema de búsqueda de caminos con sólo parcial

conocimiento. Por ejemplo, sospechamos qué bordes podrían reenviar un pago porque su capacidad parece lo suficientemente grande. Pero no podemos estar seguros a menos que lo probemos o preguntemos directamente a los propietarios del canal. Incluso si pudiéramos preguntar directamente a los propietarios del canal, su saldo podría cambiar en el momento en que hayamos preguntado a otros, calculado una ruta, construido una cebolla y enviado. No solo tenemos información limitada, sino que la información que tenemos es muy dinámica y puede cambiar en cualquier momento sin nuestro conocimiento.

Manteniéndolo simple

El mecanismo de búsqueda de rutas implementado en los nodos Lightning consiste en crear primero una lista de rutas candidatas, filtradas y ordenadas por alguna función. Luego, el nodo o la billetera probará las rutas (al intentar entregar un pago) en un ciclo de prueba y error hasta que se encuentre una ruta que entregue el pago con éxito.

> **NOTA**
>
> Este sondeo lo realiza el nodo Lightning o la billetera y no lo observa directamente el usuario del software. Sin embargo, el usuario puede sospechar que se está realizando un sondeo si el pago no se completa al instante.

Si bien el sondeo ciego no es óptimo y deja un amplio margen de mejora, se debe tener en cuenta que incluso esta estrategia simplista funciona sorprendentemente bien para pagos más pequeños y nodos bien conectados.

La mayoría de las implementaciones de billeteras y nodos Lightning mejoran este enfoque al ordenar/ponderar la lista de rutas candidatas. Algunas implementaciones ordenan las rutas candidatas por costo (tarifas) o alguna combinación de costo y capacidad.

Proceso de entrega de pago y búsqueda de ruta

La búsqueda de ruta y la entrega del pago implican varios pasos, que enumeramos aquí. Diferentes implementaciones pueden usar diferentes algoritmos y estrategias, pero es probable que los pasos básicos sean muy similares:

1. Crea un ***gráfico de canal*** a partir de anuncios y actualizaciones que contiene la capacidad de cada canal, y filtrar el gráfico, ignorando aquellos canales con capacidad insuficiente para la cantidad que queremos enviar.

2. Busque rutas que conecten al remitente con el destinatario.

3. Ordene las rutas por algún peso (esto puede ser parte del algoritmo del paso anterior).

4. Pruebe cada ruta en orden hasta que el pago sea exitoso (la prueba y error círculo).

5. Opcionalmente, use los retornos de falla de HTLC para actualizar nuestro gráfico, reducción de la incertidumbre.

Podemos agrupar estos pasos en tres actividades principales:

- Construcción de gráfico de canal
- Pathfinding (filtrado y ordenado por algunas heurísticas)
- Intento(s) de pago

Estas tres actividades se pueden repetir en una ***ronda de pago*** si utilizamos las devoluciones fallidas para actualizar la gráfica, o si estamos haciendo pagos multiparte (ver “Pagos Multiparte”).

En las siguientes secciones veremos cada uno de estos pasos con más detalle, así como estrategias de pago más avanzadas.

Construcción de gráfico de canal

En el Capítulo 11 cubrimos los tres mensajes principales que los nodos usan en sus chismes: nodo_anuncio, canal_anuncio y

canal_actualización. Estos tres mensajes permiten que cualquier nodo construya gradualmente un "mapa" de Lightning Network en forma de un ***gráfico de canal.*** Cada uno de estos mensajes proporciona información fundamental para el gráfico de canales:

anuncio_nodo

Contiene la información sobre un nodo en Lightning Network, como su ID de nodo (clave pública), dirección de red (por ejemplo, IPv4/6 o Tor), capacidades/características, etc.

canal_anuncio

Contiene la capacidad y el ID de canal de un canal público (anunciado) entre dos nodos y prueba de la existencia y propiedad del canal.

canal_actualizar

Contiene las expectativas de tarifa y bloqueo de tiempo (CLTV) de un nodo para enrutar un pago saliente (desde la perspectiva de ese nodo) a través de un canal específico.

En términos de un gráfico matemático, el anuncio de nodo es la información necesaria para crear los nodos o ***vértices*** del gráfico. El channel_announcement nos permite crear los ***bordes*** del gráfico que representan los canales de pago. Dado que cada dirección del canal de pago tiene su propio saldo, creamos un gráfico dirigido. El channel_update nos permite incorporar tarifas y bloqueos de tiempo para establecer el ***costo*** o el ***peso*** de los bordes del gráfico.

Dependiendo del algoritmo que usaremos para la búsqueda de rutas, podemos establecer una serie de funciones de costo diferentes para los bordes del gráfico.

Por ahora, ignoremos la función de costo y simplemente establezcamos un gráfico de canal que muestre nodos y canales, usando los mensajes node_announcement y channel_announcement.

En este capítulo veremos cómo Selena intenta encontrar un camino para pagarle a Rashid un millón de satoshis. Para comenzar, Selena está construyendo un gráfico de canales usando la información de chismes de Lightning Network para descubrir nodos y canales. Luego, Selena explorará el gráfico de su canal para encontrar una ruta para enviar un pago a Rashid.

Este es el gráfico de canales ***de Selena .*** No existe ***el*** gráfico de canales, solo existe ***un gráfico de canales,*** y siempre desde la perspectiva del nodo que lo ha construido (ver “La relación mapa-territorio”).

PROPINA

Selena no construye un gráfico de canal solo cuando envía un pago. Más bien, el nodo de Selena construye y actualiza ***continuamente*** un gráfico de canal. Desde el momento en que el nodo de Selena se inicia y se conecta a cualquier par en la red, participará en los chismes y usará cada mensaje para aprender tanto como sea posible sobre la red.

LA RELACIÓN MAPA-TERRITORIO

De la página de Wikipedia sobre la Relación Mapa-Territorio, “La relación mapa-territorio describe la relación entre un objeto y una representación de ese objeto, como en la relación entre un territorio geográfico y un mapa del mismo.”

La relación mapa-territorio se ilustra mejor en "Sylvie and Bruno Concluded", una historia corta de Lewis Carroll que describe un mapa ficticio que es una escala 1: 1 del territorio que mapea, por lo que tiene una precisión perfecta pero se vuelve completamente inútil. cubriría todo el territorio si se desplegara.

¿Qué significa esto para Lightning Network? Lightning Network es el territorio, y un gráfico de canal es un mapa de ese territorio.

Si bien podríamos imaginar un gráfico de canal teórico (ideal platónico) que represente el mapa completo y actualizado de Lightning Network, dicho mapa es simplemente Lightning Network en sí. ¡Cada nodo tiene su propio gráfico de canal que se construye a partir de anuncios y es necesariamente incompleto, incorrecto y desactualizado!

El mapa nunca puede describir de forma completa y precisa el territorio.

Selena escucha los mensajes de anuncio de nodo y descubre otros cuatro nodos (además de Rashid, el destinatario previsto). El gráfico resultante representa una red de seis nodos: Selena y Rashid son el remitente y el destinatario, respectivamente; Alice, Bob, Xavier y Yan son nodos intermediarios. El gráfico inicial de Selena es solo una lista de nodos, que se muestra en la figura 12-3.

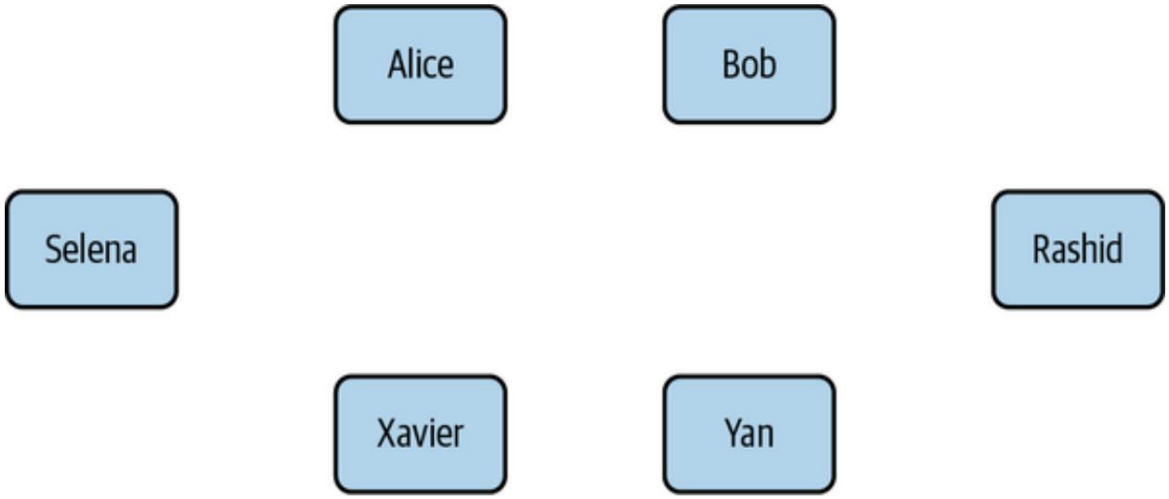

Figura 12-3. Anuncios de nodos

Selena también recibe siete mensajes de anuncio de canal con las capacidades de canal correspondientes, lo que le permite construir un "mapa" básico de la red, que se muestra en la Figura 12-4. (Los nombres Alice, Bob, Selena, Xavier, Yan y Rashid han sido reemplazados por sus iniciales: A, B, S, X y R, respectivamente).

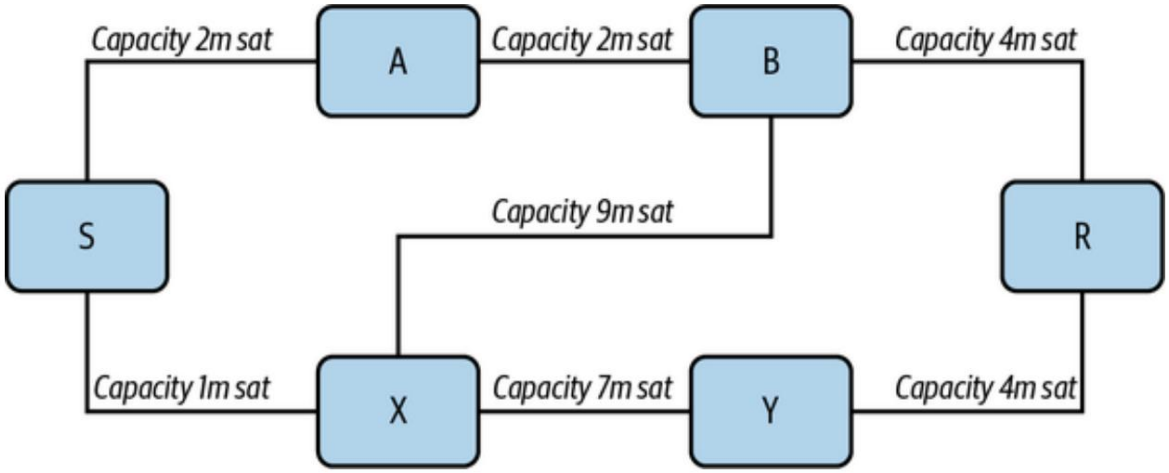

Figura 12-4. El gráfico del canal

Incertidumbre en el gráfico de canales

Como puede ver en la Figura 12-4, Selena no conoce ninguno de los saldos de los canales. Su gráfico de canal inicial contiene el nivel más alto de incertidumbre.

Pero espera: ¡Selena conoce ***algunos*** balances de canales! Conoce los saldos de los canales que su propio nodo ha conectado con otros nodos.

Si bien esto no parece mucho, de hecho es información muy importante para construir un camino: Selena conoce la liquidez real de sus propios canales. Actualicemos el gráfico del canal para mostrar esta información. Usaremos un "?" símbolo para representar los saldos desconocidos, como se muestra en la figura 12-5.

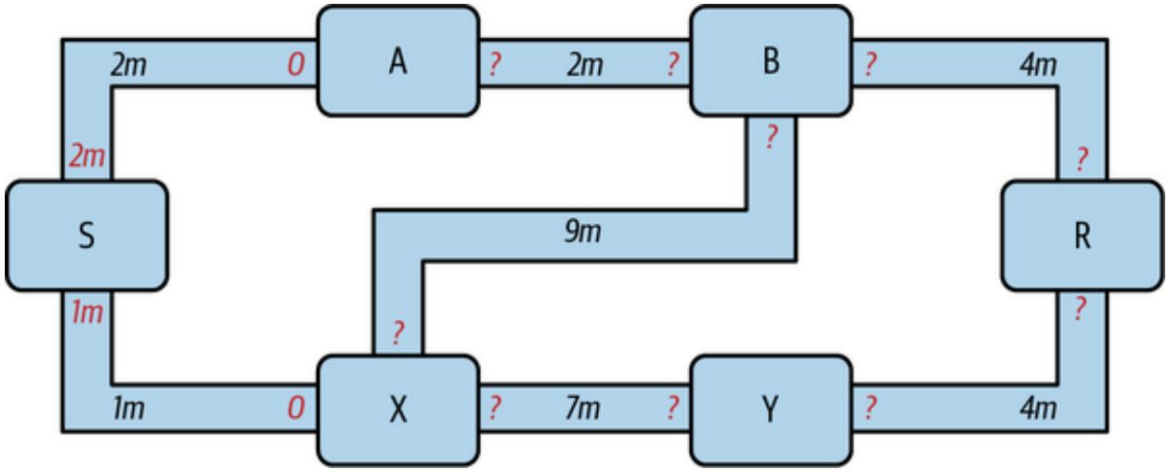

Figura 12-5. Gráfico de canal con saldos conocidos y desconocidos

Mientras que la "?" símbolo parece ominoso, la falta de certeza no es lo mismo que la ignorancia completa. Podemos ***cuantificar*** la incertidumbre y ***reducirla*** actualizando el gráfico con los HTLC exitosos/fallidos que intentamos.

La incertidumbre se puede cuantificar, porque conocemos la liquidez máxima y mínima posible y podemos calcular probabilidades para rangos más pequeños (más precisos).

Una vez que intentamos enviar un HTLC, podemos obtener más información sobre los saldos del canal: si tenemos éxito, entonces el saldo fue ***al menos*** suficiente para transportar la cantidad específica. Mientras tanto, si recibimos un error de "falla temporal del canal", la razón más probable es la falta de liquidez para la cantidad específica.

PROPINA

Puede estar pensando: "¿Cuál es el punto de aprender de un HTLC exitoso?" Después de todo, si tuvo éxito, estamos "terminados". Pero considere que podemos estar enviando una parte de un pago de varias partes. También es posible que enviemos otros pagos de una sola parte dentro de poco tiempo. ¡Todo lo que aprendamos sobre la liquidez es útil para el próximo intento!

Liquidez Incertidumbre y Probabilidad

Para cuantificar la incertidumbre de la liquidez de un canal, podemos aplicar la teoría de la probabilidad. Un modelo básico de la probabilidad de entrega del pago conducirá a algunas conclusiones bastante obvias, pero importantes:

- Los pagos más pequeños tienen una mejor oportunidad de entrega exitosa a lo largo de una ruta.
- Los canales de mayor capacidad nos darán una mejor oportunidad de entrega de pagos por un monto específico.
- Cuantos más canales (saltos), menor será la probabilidad de éxito.

Si bien estos pueden ser obvios, tienen implicaciones importantes, especialmente para el uso de pagos de varias partes (consulte “Pagos de varias partes”). Las matemáticas no son difíciles de seguir.

Usemos la teoría de la probabilidad para ver cómo llegamos a estas conclusiones.

Primero, supongamos que un canal con capacidad ***c*** tiene liquidez en un lado con un valor desconocido en el rango de (0, c) o "rango entre 0 y c". Por ejemplo, si la capacidad es 5, entonces la liquidez estará en el rango (0, 5).
Ahora bien, de esto vemos que si queremos enviar 5 satoshis, nuestra probabilidad de éxito es solo de 1 en 6 (16,66%), porque solo tendremos éxito si la liquidez es exactamente 5.

Más simplemente, si los valores posibles para la liquidez son 0, 1, 2, 3, 4 y 5, solo uno de esos seis valores posibles será suficiente para enviar nuestro pago.
Para continuar con este ejemplo, si el monto de nuestro pago fuera 3, tendríamos éxito si la liquidez fuera 3, 4 o 5. Por lo tanto, nuestras posibilidades de éxito son 3 en 6

(50%). Expresado en matemáticas, la función de probabilidad de éxito para un solo canal es:

$$PC(a) = (c + 1 \ddot{y} a)/ (c + 1)$$

donde ***a*** es la cantidad y ***c*** es la capacidad.

De la ecuación vemos que si la cantidad es cercana a 0, la probabilidad es cercana a 1, mientras que si la cantidad excede la capacidad, la probabilidad es cero.

En otras palabras: "Los pagos más pequeños tienen más posibilidades de entrega exitosa" o "Los canales de mayor capacidad nos brindan mejores posibilidades de entrega por un monto específico" y "No puede enviar un pago en un canal con capacidad insuficiente".

Ahora pensemos en la probabilidad de éxito a través de un camino formado por varios canales. Digamos que nuestro primer canal tiene un 50 % de posibilidades de éxito ***(P*** = 0,5). Entonces, si nuestro segundo canal tiene un 50 % de posibilidades de éxito ***(P*** = 0,5), es intuitivo que nuestra posibilidad general es del 25 % ***(P*** = 0,25).

Podemos expresar esto como una ecuación que calcula la probabilidad de éxito de un pago como el producto de las probabilidades para cada canal en la(s) ruta(s):

$$P_{pago} = \ddot{y}_{i=1}^{n} P_i$$

Donde ***P*** es la probabilidad de éxito en un camino o canal, y ***pago p*** es la probabilidad global de un pago exitoso sobre todos los caminos/canales.

De la ecuación vemos que dado que la probabilidad de éxito en un solo canal siempre es menor o igual a 1, la probabilidad en muchos canales ***caerá exponencialmente.***

En otras palabras, “Cuantos más canales (saltos) utilice, menor será la probabilidad de éxito”.

> **NOTA**
>
> Hay mucha teoría matemática y modelado detrás de la incertidumbre de la liquidez en los canales. El trabajo fundamental sobre el modelado de los intervalos de incertidumbre de la liquidez del canal se puede encontrar en el documento "Seguridad y privacidad de los pagos de Lightning Network con saldos de canal inciertos" por (coautor de este libro) Pickhardt et al.

Tarifas y otras métricas del canal

A continuación, nuestro remitente agregará información al gráfico de los mensajes channel_update recibidos de los nodos intermediarios. Como recordatorio, channel_update contiene una gran cantidad de información sobre un canal y las expectativas de uno de los socios del canal.

En la Figura 12-6 vemos cómo Selena puede actualizar el gráfico del canal en función de los mensajes channel_update de A, B, X e Y. Tenga en cuenta que el ID del canal y la dirección del canal (incluidos en channel_flags) le indican a Selena qué canal y en qué dirección esta actualización. se refiere a. Cada socio de canal chismea uno o más mensajes channel_update para anunciar sus expectativas de tarifas y otra información sobre el canal. Por ejemplo, en la parte superior izquierda vemos channel_update enviado por Alice para el canal A—B y la dirección A-to-B. Con esta actualización, Alice le dice a la red cuánto cobrará en tarifas para enrutar un HTLC a Bob a través de ese canal específico. Bob puede anunciar una actualización de canal (que no se muestra en este diagrama) para la dirección opuesta con expectativas de tarifas completamente diferentes. Cualquier nodo puede enviar una nueva actualización de canal para cambiar las tarifas o las expectativas de bloqueo de tiempo en cualquier momento.

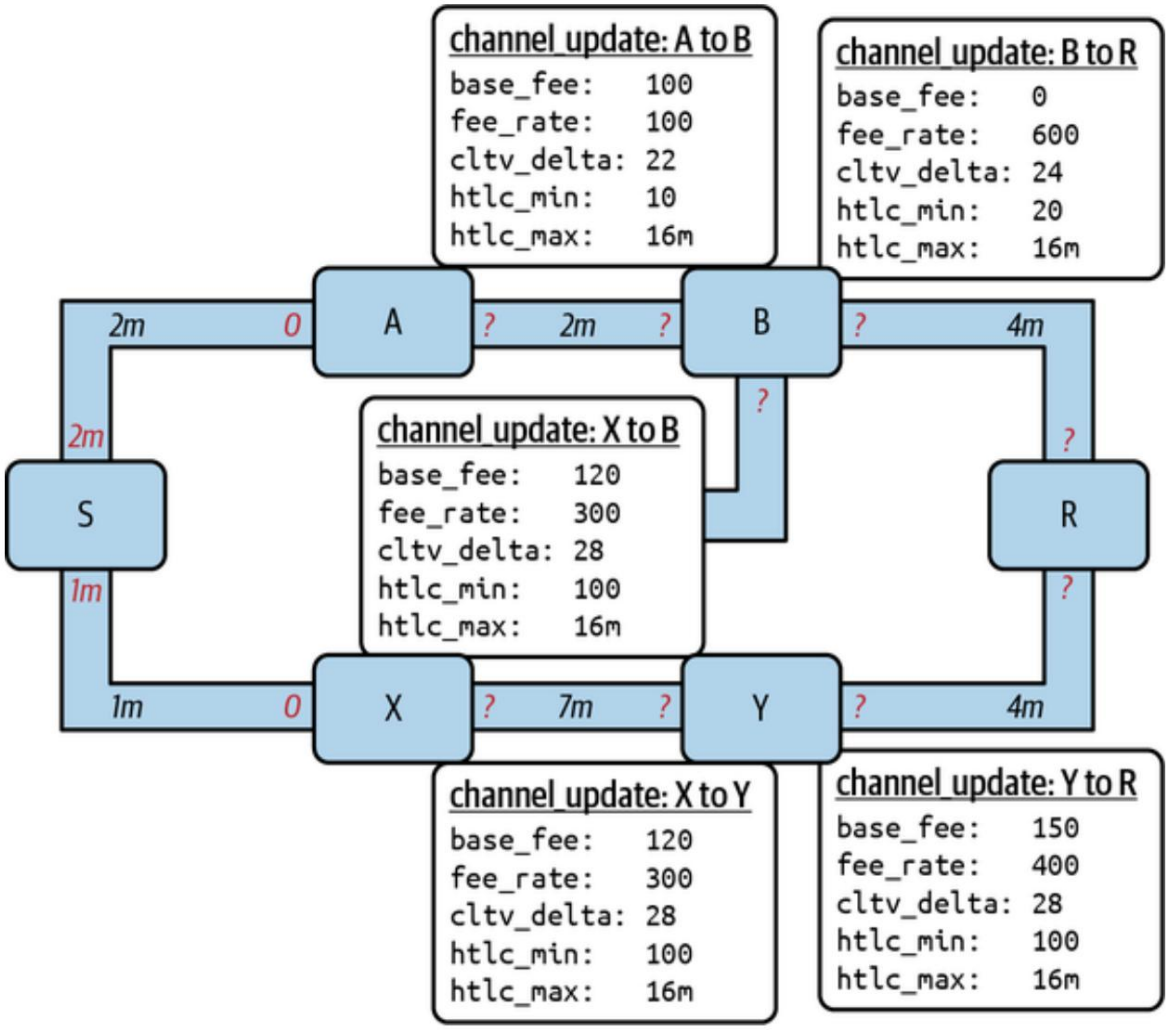

Figura 12-6. Tarifas de gráficos de canales y otras métricas de canales

La información de tarifa y bloqueo de tiempo es muy importante, no solo como métrica de selección de ruta. Como vimos en el Capítulo 10, el remitente debe sumar tarifas y plazos (cltv_expiry_delta) en cada salto para hacer la cebolla.
El proceso de cálculo de tarifas ocurre desde el destinatario hasta el remitente ***hacia atrás*** a lo largo de la ruta porque cada salto intermediario espera un HTLC entrante con una cantidad mayor y un tiempo de caducidad mayor que el HTLC saliente que enviarán al siguiente salto. Entonces, por ejemplo, si Bob quiere 1,000 satoshis en tarifas y 30 bloques de delta de bloqueo de tiempo de vencimiento para enviar un pago a Rashid, entonces esa cantidad y el delta de vencimiento deben agregarse al HTLC ***de Alice.***

También es importante tener en cuenta que un canal debe tener liquidez suficiente no solo para el monto del pago sino también para las tarifas acumuladas.

de todos los saltos posteriores. Aunque el canal de Selena a Xavier (SÿX) tiene suficiente liquidez para un pago de satoshi de 1 millón, ***no*** tiene suficiente liquidez una vez que consideramos las tarifas. Necesitamos conocer las tarifas porque solo se considerarán las rutas que tengan suficiente liquidez ***tanto para el pago como para todas las tarifas .***

Encontrar rutas de candidatos

Encontrar una ruta adecuada a través de un gráfico dirigido como este es un problema informático bien estudiado (conocido ampliamente como el ***problema de la ruta más corta),*** que puede resolverse mediante una variedad de algoritmos según la optimización deseada y las restricciones de recursos.

El algoritmo más famoso para resolver este problema fue inventado por el matemático holandés EW Dijkstra en 1956, conocido simplemente como el ***algoritmo de Dijkstra .*** Además del algoritmo original de Dijkstra, hay muchas variaciones y optimizaciones, como A* ("A-estrella"), que es un algoritmo basado en heurística.

Como se mencionó anteriormente, la "búsqueda" debe aplicarse ***hacia atrás*** para tener en cuenta las tarifas que se acumulan desde el destinatario hasta el remitente. Por lo tanto, Dijkstra, A* o algún otro algoritmo buscaría una ruta desde el destinatario hasta el remitente, utilizando tarifas, liquidez estimada y delta de bloqueo de tiempo (o alguna combinación) como una función de costo para cada salto.

Usando uno de esos algoritmos, Selena calcula varios caminos posibles para Rashid, ordenados por la ruta más corta:

1. SÿAÿBÿR
2. SÿXÿYÿR
3. SÿXÿBÿR
4. SÿAÿBÿXÿYÿR

Pero, como vimos anteriormente, el canal SÿX no tiene suficiente liquidez para un pago de satoshi de 1 millón una vez que se consideran las tarifas. Entonces las Rutas 2 y

3 no son viables. Eso deja las Rutas 1 y 4 como posibles rutas para el pago.

¡Con dos caminos posibles, Selena está lista para intentar la entrega!

Entrega de pago (bucle de prueba y error)

El nodo de Selena inicia el ciclo de prueba y error construyendo los HTLC, construyendo la cebolla e intentando entregar el pago. Para cada intento, hay tres resultados posibles:

- Un resultado exitoso (update_fulfill_htlc)
- Un error (update_fail_htlc)
- Un pago "atascado" sin respuesta (ni éxito ni fracaso)

Si el pago falla, se puede volver a intentar a través de una ruta diferente actualizando el gráfico (cambiando las métricas de un canal) y recalculando una ruta alternativa.

Vimos lo que sucede si el pago está "atascado" en "Pagos atascados".
El detalle importante es que un pago atascado es el peor resultado porque no podemos volver a intentarlo con otro HTLC ya que ambos (el atascado y el reintento) podrían pasar eventualmente y causar un pago doble.

Primer intento (Ruta #1)

Selena intenta el primer camino (SÿAÿBÿR). Ella construye la cebolla y la envía, pero recibe un código de error del nodo de Bob. Bob informa una falla temporal del canal con una actualización de canal que identifica el canal BÿR como el que no puede entregar. Este intento se muestra en la figura 12-7.

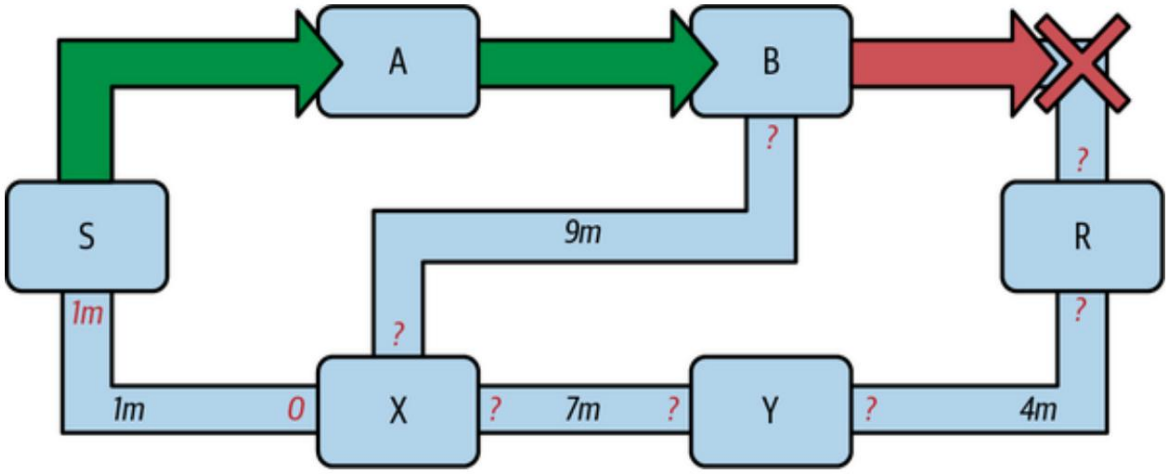

Figura 12-7. El intento de la ruta n.º 1 falla

Aprendiendo del fracaso

A partir de este código de falla, Selena deducirá que Bob no tiene suficiente liquidez para entregar el pago a Rashid en ese canal. ¡Es importante destacar que esta falla reduce la incertidumbre de la liquidez de ese canal! Anteriormente, el nodo de Selena asumía que la liquidez en el lado del canal de Bob estaba en algún lugar en el rango (0, 4M). Ahora, puede asumir que la liquidez está en el rango (0, 999999). De manera similar, Selena ahora puede suponer que la liquidez de ese canal del lado de Rashid está en el rango (1M, 4M), en lugar de (0, 4M). Selena ha aprendido mucho de este fracaso.

Segundo intento (Ruta #4)

Ahora Selena intenta el cuarto camino candidato (SÿAÿBÿXÿYÿR).
Esta es una ruta más larga e incurrirá en más tarifas, pero ahora es la mejor opción para la entrega del pago.

Afortunadamente, Selena recibe un mensaje update_fulfill_htlc de Alice, que indica que el pago se realizó correctamente, como se muestra en la Figura 12-8.

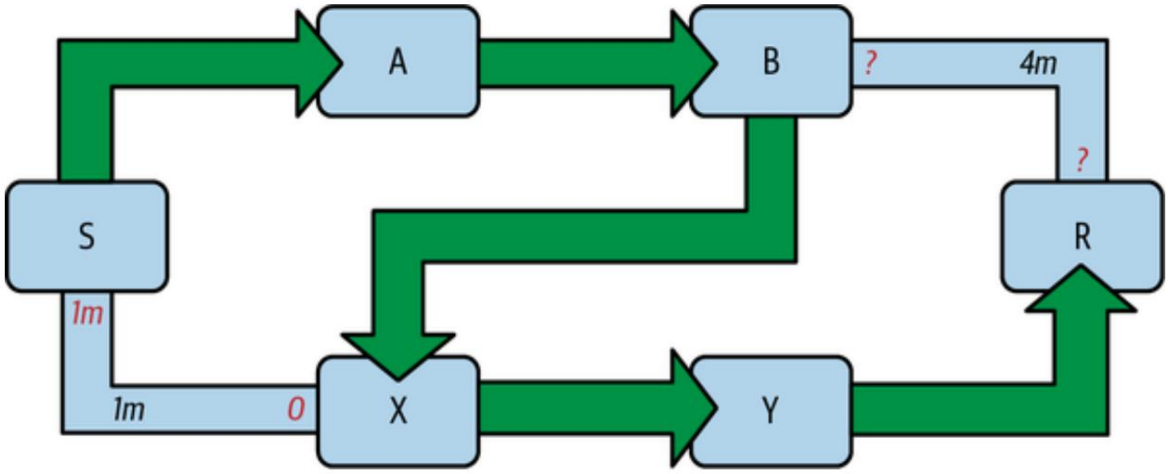

Figura 12-8. El intento de la ruta n.º 4 tiene éxito

Aprendiendo del éxito

Selena también ha aprendido mucho de este exitoso pago. Ahora sabe que todos los canales en el camino SÿAÿBÿXÿYÿR tenían suficiente liquidez para entregar el pago. Además, ahora sabe que cada uno de estos canales ha movido la cantidad de HTLC (1 millón + tarifas) al otro extremo del canal. Esto le permite a Selena recalcular el rango de liquidez en el lado receptor de todos los canales en ese camino, reemplazando la liquidez mínima con 1M + tarifas.

¿Conocimiento obsoleto?

Selena ahora tiene un "mapa" mucho mejor de Lightning Network (al menos en lo que respecta a estos siete canales). Este conocimiento será útil para cualquier pago posterior que Selena intente realizar.

Sin embargo, este conocimiento se vuelve algo "obsoleto" a medida que los otros nodos envían o enrutan pagos. Selena nunca verá ninguno de estos pagos (a menos que ella sea el remitente). Incluso si ella está involucrada en el enrutamiento de pagos, el mecanismo de enrutamiento de cebolla significa que solo puede ver los cambios para un salto (sus propios canales).

Por lo tanto, el nodo de Selena debe considerar cuánto tiempo conservar este conocimiento antes de asumir que está obsoleto y ya no es útil.

Pagos de varias partes

Los pagos multiparte (MPP) son una función que se introdujo en Lightning Network en 2020 y ya están ampliamente disponibles. Los pagos de varias partes permiten que un pago se divida en varias ***partes*** que se envían como HTLC a través de varias rutas diferentes al destinatario previsto, preservando la ***atomicidad*** del pago total. En este contexto, la atomicidad significa que todas las partes de HTLC de un pago finalmente se cumplen o el pago completo falla y todas las partes de HTLC fallan. No hay posibilidad de un pago parcialmente exitoso.

Los pagos de varias partes son una mejora significativa en Lightning Network porque permiten enviar montos que no "encajarán" en ningún canal individual al dividirlos en montos más pequeños para los cuales hay suficiente liquidez. Además, se ha demostrado que los pagos de varias partes aumentan la probabilidad de un pago exitoso, en comparación con un pago de ruta única.

PROPINA

Ahora que el MPP está disponible, es mejor pensar en un pago de ruta única como una subcategoría de un MPP. Esencialmente, una ruta única es solo una parte múltiple de tamaño uno. Todos los pagos pueden considerarse pagos de varias partes, a menos que el tamaño del pago y la liquidez disponible hagan posible la entrega con una sola parte.

Usando MPP

MPP no es algo que un usuario seleccionará, sino que es una estrategia de entrega de pagos y búsqueda de rutas de nodos. Se implementan los mismos pasos básicos: crear un gráfico, seleccionar rutas y el ciclo de prueba y error. La diferencia es que durante la selección de ruta también debemos considerar cómo dividir el pago para optimizar la entrega.

En nuestro ejemplo, podemos ver algunas mejoras inmediatas a nuestro problema de búsqueda de rutas que son posibles con MPP. Primero, podemos utilizar el canal SÿX que tiene liquidez insuficiente para transportar 1 millón de satoshis más tarifas. Al enviar una parte más pequeña a lo largo de ese canal, podemos usar

caminos que antes no estaban disponibles. En segundo lugar, tenemos la liquidez desconocida del canal BÿR, que es insuficiente para transportar la cantidad de 1M, pero podría ser suficiente para transportar una cantidad menor.

Fraccionamiento de pagos

La cuestión fundamental es cómo dividir los pagos. Más específicamente, ¿cuál es el número óptimo de divisiones y las cantidades óptimas para cada división?

Esta es un área de investigación en curso donde están surgiendo nuevas estrategias. Los pagos de varias partes conducen a un enfoque algorítmico diferente al de los pagos de una sola ruta, aunque las soluciones de una sola ruta pueden surgir de una optimización de varias partes (es decir, una sola ruta puede ser la solución óptima sugerida por un algoritmo de búsqueda de rutas de varias partes).

Si recuerda, encontramos que la incertidumbre de la liquidez/saldos lleva a algunas conclusiones (algo obvias) que podemos aplicar en la búsqueda de ruta de MPP, a saber:

- Los pagos más pequeños tienen una mayor probabilidad de éxito.
- Cuantos más canales utilice, la probabilidad de éxito se vuelve (exponencialmente) menor.

A partir de la primera de estas ideas, podríamos concluir que dividir un pago grande (por ejemplo, 1 millón de satoshis) en pagos pequeños aumenta la posibilidad de que cada uno de esos pagos pequeños tenga éxito. El número de caminos posibles con suficiente liquidez será mayor si enviamos cantidades más pequeñas.

Para llevar esta idea al extremo, ¿por qué no dividir el pago de 1 millón de satoshi en un millón de partes separadas de un satoshi? Bueno, la respuesta está en nuestra segunda idea: dado que usaríamos más canales/rutas para enviar nuestro millón de HTLC de un solo satoshi, nuestra probabilidad de éxito se reduciría exponencialmente.

Si no es obvio, las dos ideas anteriores crean un "punto óptimo" en el que podemos maximizar nuestras posibilidades de éxito: dividir en pagos más pequeños, ¡pero no en demasiadas divisiones!

Cuantificar este balance óptimo de tamaño/número de divisiones para un gráfico de canal dado está fuera del alcance de este libro, pero es un área activa de investigación. Algunas implementaciones actuales utilizan una estrategia muy simple de dividir el pago en dos mitades, cuatro cuartos, etc.

> **NOTA**
>
> Para obtener más información sobre el problema de optimización conocido como flujos de costo mínimo involucrados al dividir los pagos en diferentes tamaños y asignarlos a rutas, consulte el artículo "Flujos de pago óptimos y económicos en Lightning Network" por (coautor de este libro) René Pickhardt y Stefan Richter.

En nuestro ejemplo, el nodo de Selena intentará dividir el pago de 1 millón de satoshi en 2 partes con 600k y 400k satoshi, respectivamente, y enviarlos por 2 rutas diferentes. Esto se muestra en la Figura 12-9.

Debido a que ahora se puede utilizar el canal SÿX y (afortunadamente para Selena), el canal BÿR tiene suficiente liquidez para 600k satoshis, las 2 partes tienen éxito en caminos que antes no eran posibles.

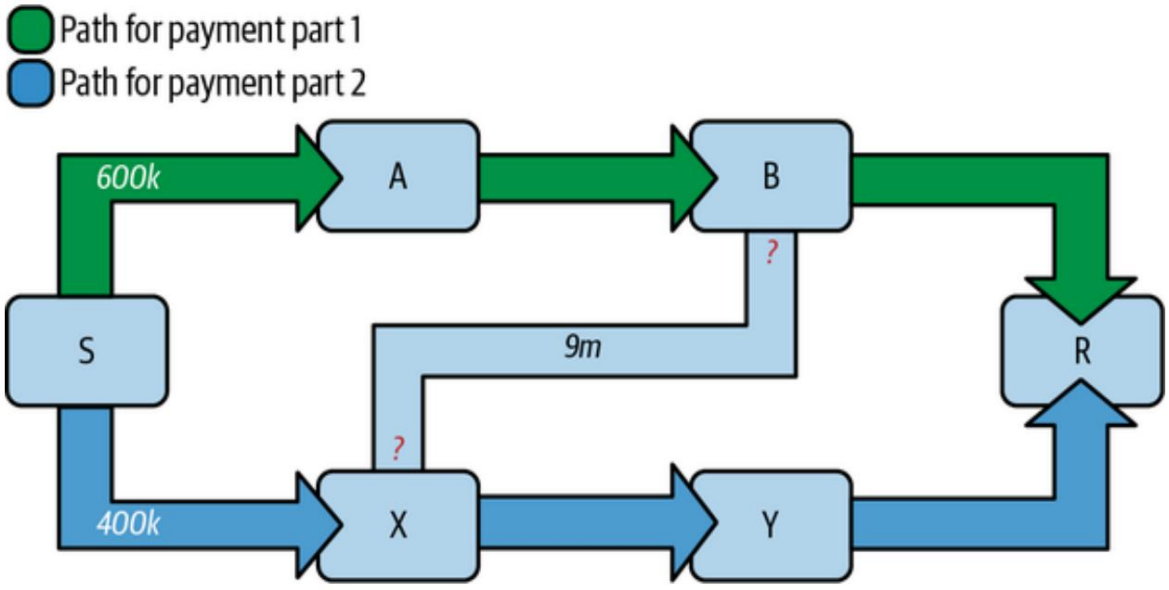

Figura 12-9. Envío de dos partes de un pago de varias partes

Prueba y error en múltiples "Rondas"

Los pagos de varias partes conducen a un ciclo de prueba y error algo modificado para la entrega del pago. Debido a que estamos intentando múltiples caminos en cada intento, tenemos cuatro resultados posibles:

- Todas las partes son exitosas, el pago es exitoso
- Algunas partes tienen éxito, otras fallan con errores devueltos
- Todas las partes fallan con errores devueltos
- Algunas partes están "atascadas", no se devuelven errores

En el segundo caso, donde algunas partes fallan con errores devueltos y algunas partes tienen éxito, ahora podemos ***repetir*** el ciclo de prueba y error, pero ***solo por la cantidad residual.***

Supongamos, por ejemplo, que Selena tenía un gráfico de canales mucho más grande con cientos de caminos posibles para llegar a Rashid. Su algoritmo de búsqueda de rutas podría encontrar una división de pago óptima que consta de 26 partes de diferentes tamaños.
Después de intentar enviar las 26 partes en la primera ronda, 3 de esas partes fallaron con errores.

Si esas 3 partes consistieran en, digamos, 155k satoshis, entonces Selena reiniciaría el esfuerzo de búsqueda de caminos, solo para 155k satoshis. ¡La próxima ronda podría encontrar caminos completamente diferentes (optimizados para la cantidad residual de 155k) y dividir la cantidad de 155k en divisiones completamente diferentes!

PROPINA

Si bien parece que 26 partes divididas son muchas, las pruebas en Lightning Network han entregado con éxito un pago de 0.3679 BTC al dividirlo en 345 partes.

Además, el nodo de Selena actualizaría el gráfico del canal utilizando la información recopilada de los éxitos y errores de la primera ronda para encontrar las rutas y divisiones más óptimas para la segunda ronda.

Digamos que el nodo de Selena calcula que la mejor manera de enviar el residuo de 155k es 6 partes divididas en 80k, 42k, 15k, 11k, 6.5k y 500 satoshis. En el

En la siguiente ronda, Selena obtiene solo un error, lo que indica que la parte del satoshi de 11k falló. Una vez más, Selena actualiza el gráfico del canal en función de la información recopilada y vuelve a ejecutar la búsqueda de rutas para enviar el residuo de 11k. Esta vez, lo logra con 2 partes de satoshis de 6k y 5k, respectivamente.

Este ejemplo de múltiples rondas de envío de un pago usando MPP se muestra en la Figura 12-10.

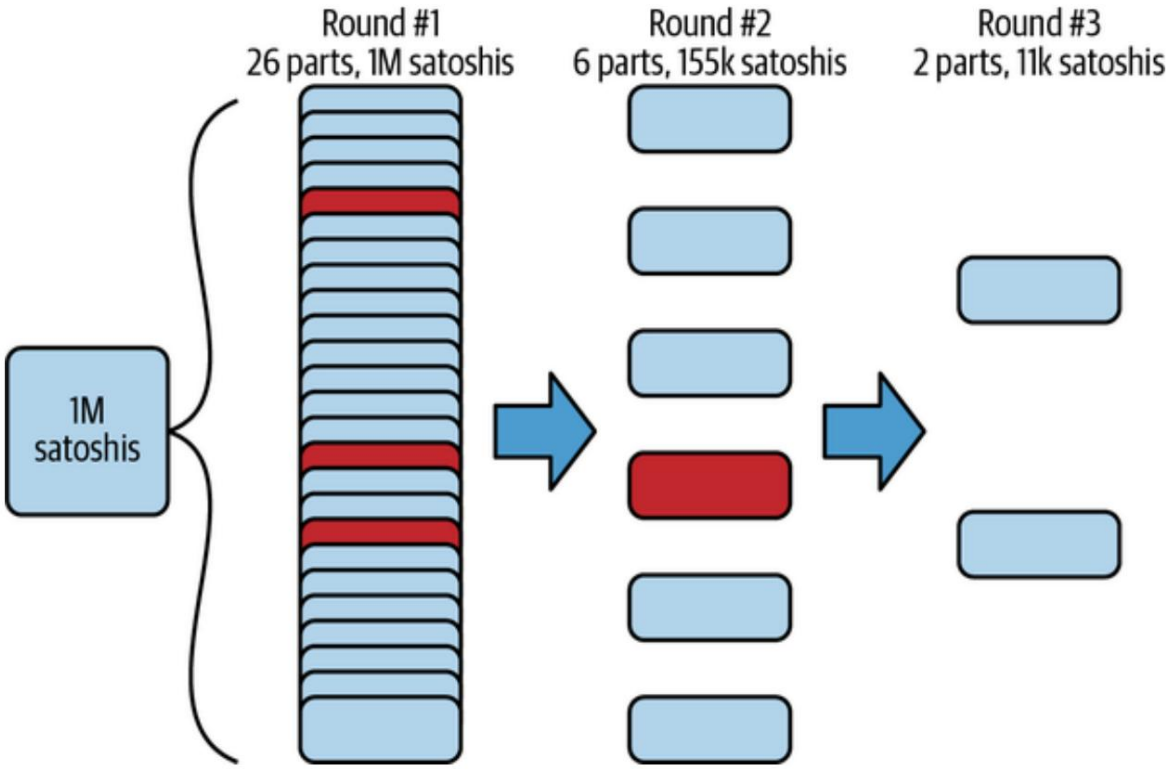

Figura 12-10. Envío de un pago en múltiples rondas con MPP

Al final, el nodo de Selena usó 3 rondas de búsqueda de rutas para enviar 1 millón de satoshis en 30 partes.

Conclusión

En este capítulo analizamos la búsqueda de rutas y la entrega de pagos. Vimos cómo usar el gráfico de canales para encontrar rutas de un remitente a un destinatario. También vimos cómo el remitente intentará entregar pagos en una ruta candidata y repetirá en un ciclo de prueba y error.

También examinamos la incertidumbre de la liquidez del canal (desde la perspectiva del remitente) y las implicaciones que tiene para la búsqueda de rutas. Vimos cómo podemos cuantificar la incertidumbre y usar la teoría de la probabilidad para sacar algunas conclusiones útiles. También vimos cómo podemos reducir la incertidumbre aprendiendo de los pagos exitosos y fallidos.

Finalmente, vimos cómo la función de pagos de varias partes recientemente implementada nos permite dividir los pagos en partes, lo que aumenta la probabilidad de éxito incluso para pagos más grandes.

Capítulo 13. Protocolo de conexión: estructura y extensibilidad

En este capítulo, nos sumergimos en el protocolo de cable de Lightning Network y también cubrimos todas las diversas palancas de extensibilidad que se han integrado en el protocolo. Al final de este capítulo, un lector ambicioso debería poder escribir su propio analizador de protocolo de cable para Lightning Network. Además de poder escribir un analizador de protocolo de cable personalizado, el lector de este capítulo obtendrá una comprensión profunda de los diversos mecanismos de actualización que se han integrado en el protocolo.

Capa de mensajería en el protocolo Lightning Suite

La capa de mensajería, que se detalla en este capítulo, consta de "Encuadre y formato de mensaje", codificación "Tipo-Longitud-Valor" y "bits de características".
Estos componentes se destacan mediante un esquema en el conjunto de protocolos, que se muestra en la Figura 13-1.

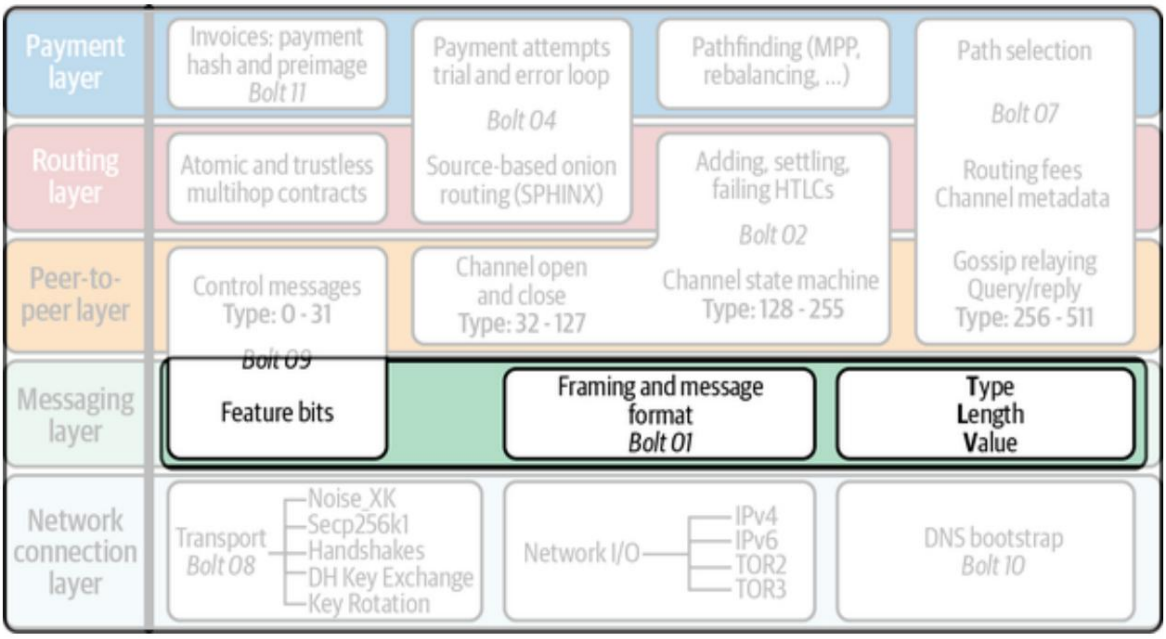

Figura 13-1. Capa de mensajería en el conjunto de protocolos Lightning

Estructura de alambre

Comenzamos describiendo la estructura de alto nivel del entramado de cables ***dentro*** del protocolo. Cuando decimos tramas, nos referimos a la forma en que los bytes se empaquetan en el cable para ***codificar*** un mensaje de protocolo particular. Sin el conocimiento del sistema de tramas utilizado en el protocolo, una cadena de bytes en el cable parecería una serie de bytes aleatorios porque no se ha impuesto una estructura. Al aplicar el marco adecuado para decodificar estos bytes en el cable, podremos extraer la estructura y finalmente analizar esta estructura en mensajes de protocolo dentro de nuestro lenguaje de nivel superior.

Es importante tener en cuenta que Lightning Network es un protocolo ***encriptado de extremo a extremo ,*** y la estructura del cable está encapsulada dentro de una capa de transporte de mensajes ***encriptados .*** Como vemos en el Capítulo 14, Lightning Network usa una variante personalizada del Noise Protocol para manejar el cifrado de transporte. Dentro de este capítulo, cada vez que damos un ejemplo de estructura cableada, asumimos que la capa de cifrado ya se ha eliminado (al decodificar) o que aún no hemos encriptado el conjunto de bytes antes de enviarlos por cable (codificación). .

Estructura de alambre de alto nivel

Dicho esto, estamos listos para describir el esquema de alto nivel utilizado para codificar mensajes en el cable:

- Los mensajes en el cable comienzan con un campo de tipo ***de 2 bytes ,*** seguido de una carga de mensaje.
- La carga útil del mensaje en sí puede tener un tamaño de hasta 65 KB.
- Todos los números enteros están codificados en big-endian (orden de red).
- Cualquier byte que siga después de un mensaje definido se puede ignorar de forma segura.

Sí, eso es todo. Como el protocolo se basa en una capa de cifrado de protocolo de transporte ***encapsulado ,*** no necesitamos una longitud explícita para cada tipo de mensaje.
Esto se debe al hecho de que el cifrado de transporte funciona a nivel de ***mensaje ,*** por lo que cuando estamos listos para decodificar el siguiente mensaje, ya conocemos la cantidad total de bytes del mensaje en sí. El uso de 2 bytes para el tipo de mensaje (codificado en big-endian) significa que el protocolo puede tener hasta 2^16 – 1 o 65ÿ535 mensajes distintos. Continuando, debido a que sabemos que todos los mensajes deben tener menos de 65 KB, esto simplifica nuestro análisis, ya que podemos usar un búfer ***de tamaño fijo*** y mantener límites estrictos en la cantidad total de memoria requerida para analizar un mensaje de cable entrante.

El último punto permite un grado de ***compatibilidad*** con versiones anteriores porque los nuevos nodos pueden proporcionar información en los mensajes de cable que los nodos más antiguos (que pueden no entenderlos) pueden ignorar de manera segura. Como veremos más adelante, esta función, combinada con un formato de extensibilidad de mensajes por cable muy flexible, permite que el protocolo también logre compatibilidad con versiones ***posteriores .***

Tipo de codificación

Con este trasfondo de alto nivel provisto, ahora comenzamos en la capa más primitiva: analizar tipos primitivos. Además de codificar números enteros, el Protocolo Lightning también permite la codificación de una amplia gama de tipos,

incluidos segmentos de bytes de longitud variable, claves públicas de curva elíptica, direcciones de Bitcoin y firmas. Cuando describimos la ***estructura*** de los mensajes electrónicos más adelante en este capítulo, nos referimos al tipo de alto nivel (el tipo abstracto) en lugar de la representación de nivel inferior de dicho tipo. En esta sección, eliminamos esta capa de abstracción para garantizar que nuestro futuro analizador de cables pueda codificar/decodificar correctamente cualquiera de los tipos de nivel superior.

En la tabla 13-1, asignamos el nombre de un tipo de mensaje dado a la rutina de alto nivel utilizada para codificar/decodificar el tipo.

Encuadre de tipo de alto nivel		Comentario
node_alias	Un segmento de bytes de longitud fija de 32 bytes	Al decodificar, rechazar si el contenido no es válido Cadena UTF-8
channel_id	Un segmento de bytes de longitud fija de 32 bytes que asigna un punto de salida a un valor de 32 bytes	Dado un punto de salida, uno puede convertirlo en un canal I_id tomando el TxID del punto de salida y XORing con el índice (interpretado como los 2 bytes inferiores)
short_chan_id	Un entero de 64 bits sin signo (uint64)	Compuesto por la altura del bloque (24 bits), el índice de transacción (24 bits) y el índice de salida (16 bits) empaquetados en 8 bytes
milli_satoshi	Un entero de 64 bits sin signo (uint64)	Representa la milésima parte de un satoshi
satoshi	Un entero de 64 bits sin signo (uint64)	La unidad base de bitcoin
pubkey	Una clave pública secp256k1 codificada en formato ***comprimido ,*** ocupando 33 bytes	Ocupa una longitud fija de 33 bytes en el cable
ellos mismos	Una firma ECDSA de la elíptica secp256k1 curva	Codificado como un segmento de bytes ***fijo*** de 64 bytes, empaquetado como R || S
uint8	Un entero de 8 bits	
uint16	Un entero de 16 bits	
uint64	Un entero de 64 bits	
[]byte	Un segmento de bytes de longitud variable	Con el prefijo de un entero de 16 bits que indica la longitud de los bytes
color_rgb	Codificación de colores RGB	Codificado como una serie de enteros de 8 bits
net_addr	La codificación de una dirección de red.	Codificado con un prefijo de 1 byte que indica el tipo de dirección, seguido del cuerpo de la dirección

En la siguiente sección, describimos la estructura de cada mensaje electrónico, incluido el tipo de prefijo del mensaje junto con el contenido de su cuerpo.

Extensiones de mensaje de tipo-longitud-valor

Anteriormente en este capítulo mencionamos que los mensajes pueden tener un tamaño de hasta 65 KB, y si al analizar un mensaje quedan bytes adicionales, entonces esos bytes deben ignorarse. A primera vista, este requisito puede parecer algo arbitrario; sin embargo, este requisito permite una evolución desincronizada desacoplada del propio Protocolo Lightning. Discutiremos esto más hacia el final del capítulo. Pero primero, dirigimos nuestra atención a exactamente para qué se pueden usar esos "bytes adicionales" al final de un mensaje.

El formato de mensaje de los búferes de protocolo

El formato de serialización de mensajes de Protocol Buffers (Protobuf) comenzó como un formato interno utilizado en Google y se ha convertido en uno de los formatos de serialización de mensajes más populares utilizados por los desarrolladores a nivel mundial. El formato Protobuf describe cómo se codifica un mensaje (generalmente algún tipo de estructura de datos relacionada con una API) en el cable y se decodifica en el otro extremo. Existen varios "compiladores de Protobuf" en docenas de idiomas que actúan como un puente que permite que cualquier idioma codifique un Protobuf que podrá decodificar mediante una decodificación compatible en otro idioma. Tal compatibilidad de estructuras de datos entre idiomas permite una amplia gama de innovaciones porque es posible transmitir estructuras e incluso estructuras de datos escritos a través de los límites de abstracción y lenguaje.

Los protobufs también son conocidos por su flexibilidad con respecto a cómo manejan los cambios en la estructura de los mensajes subyacentes. Siempre que se respete el esquema de numeración de campos, es posible que una escritura más reciente de Protobufs incluya información dentro de un Protobuf que puede ser desconocida para los lectores más antiguos. Cuando el viejo lector se encuentra con el nuevo serializado

formato, si hay tipos/campos que no entiende, simplemente los ***ignora .*** Esto permite que los clientes antiguos y los nuevos coexistan porque todos los clientes pueden analizar una parte del formato de mensaje más reciente.

Compatibilidad hacia adelante y hacia atrás

Los protobufs son extremadamente populares entre los desarrolladores porque tienen soporte integrado para la compatibilidad tanto hacia adelante como hacia atrás. La mayoría de los desarrolladores probablemente estén familiarizados con el concepto de compatibilidad con versiones anteriores. En términos simples, el principio establece que cualquier cambio en el formato de un mensaje o API se debe realizar de manera que no rompa el soporte para clientes más antiguos.
Dentro de nuestros ejemplos anteriores de extensibilidad de Protobuf, la compatibilidad con versiones anteriores se logra al garantizar que las nuevas incorporaciones al formato Protobuf no rompan las partes conocidas de los lectores más antiguos. La compatibilidad hacia adelante, por otro lado, es igual de importante para las actualizaciones desincronizadas; sin embargo, es menos conocido. Para que un cambio sea compatible con versiones posteriores, los clientes simplemente deben ignorar cualquier información que no entiendan. Se puede decir que el mecanismo de bifurcación suave para actualizar el sistema de consenso de Bitcoin es compatible tanto hacia adelante como hacia atrás: cualquier cliente que no actualice aún puede usar Bitcoin, y si encuentra transacciones que no entiende, simplemente las ignora. ya que sus fondos no están utilizando esas nuevas funciones.

Formato de tipo-longitud-valor

Para poder actualizar los mensajes de una manera que sea compatible tanto hacia adelante como hacia atrás, además de los bits de función (más sobre esto más adelante), Lightning Network utiliza un formato de serialización de mensajes personalizado llamado simplemente Tipo-Longitud-Valor, o TLV para abreviar. . El formato se inspiró en el formato Protobuf ampliamente utilizado y toma prestados muchos conceptos al simplificar significativamente la implementación, así como el software que interactúa con el análisis de mensajes. Un lector curioso podría preguntar, "¿por qué no usar Protobufs?"

En respuesta, los desarrolladores de Lightning responderían que podemos tener lo mejor de la extensibilidad de Protobufs y al mismo tiempo tener el beneficio

de una implementación más pequeña y por lo tanto un ataque más pequeño. A partir de la versión 3.15.6, el compilador Protobuf pesa más de 656 671 líneas de código. En comparación, la implementación de LND del formato de mensaje TLV pesa solo 2,300 líneas de código (incluidas las pruebas).

Con los antecedentes necesarios presentados, ahora estamos listos para describir el formato TLV en detalle. Se dice que una extensión de mensaje TLV es un flujo de registros TLV individuales. Un solo registro TLV tiene tres componentes: el tipo de registro, la longitud del registro y, finalmente, el valor opaco del registro:

escribe

Un número entero que representa el nombre del registro que se codifica

longitud

La longitud del registro

valor

El valor opaco del registro

Tanto el tipo como la longitud se codifican mediante un número entero de tamaño variable inspirado en el número entero de tamaño variable (varint) utilizado en el protocolo P2P de Bitcoin, llamado BigSize para abreviar.

Codificación de enteros BigSize

En su forma más completa, un entero BigSize puede representar un valor de hasta 64 bits. A diferencia del formato variante de Bitcoin, el formato BigSize codifica números enteros utilizando un orden de bytes big-endian.

La variante BigSize tiene dos componentes: el discriminante y el cuerpo.
En el contexto del entero BigSize, el discriminante comunica al decodificador el tamaño del entero de tamaño variable que sigue. Recuerde que lo único de los enteros de tamaño variable es que permiten que un analizador

para usar menos bytes para codificar números enteros más pequeños que los más grandes, ahorrando espacio. La codificación de un entero BigSize sigue una de las cuatro opciones siguientes:

1. Si el valor es menor que 0xfd (253): entonces el discriminante no se usa realmente y la codificación es simplemente el entero mismo. Esto nos permite codificar números enteros muy pequeños sin sobrecarga adicional.

2. Si el valor es menor o igual a 0xffff (65535): El el discriminante se codifica como 0xfd, lo que indica que el valor que sigue es mayor que 0xfd, pero menor que 0xffff. Luego, el número se codifica como un entero de 16 bits. Incluyendo el discriminante, podemos codificar un valor mayor a 253, pero menor a 65,535 usando 3 bytes.

3. Si el valor es inferior a 0xffffffff (4294967295): El el discriminante se codifica como 0xfe. El cuerpo se codifica con un número entero de 32 bits, incluido el discriminante, y podemos codificar un valor inferior a 4.294.967.295 con 5 bytes.

4. De lo contrario, simplemente codificamos el valor como un entero de 64 bits de tamaño completo.

Restricciones de codificación TLV

Dentro del contexto de un mensaje TLV, se dice que los tipos de registros por debajo de 2^16 están ***reservados*** para uso futuro. Los tipos más allá de este rango se deben usar para extensiones de mensajes "personalizadas" utilizadas por protocolos de aplicación de nivel superior.

El valor de un registro depende del tipo. En otras palabras, puede tomar cualquier forma porque los analizadores intentarán interpretarlo según el contexto del tipo en sí.

Codificación canónica TLV

Un problema con el formato Protobuf es que las codificaciones del mismo mensaje pueden generar un conjunto de bytes completamente diferente cuando se codifica con dos versiones diferentes del compilador. Tales instancias de una codificación no canónica no son

aceptable dentro del contexto de Lightning, ya que muchos mensajes contienen una firma del resumen del mensaje. Si es posible que un mensaje se codifique de dos maneras diferentes, entonces sería posible romper la autenticación de una firma sin darse cuenta al volver a codificar un mensaje usando un conjunto ligeramente diferente de bytes en el cable.

Para garantizar que todos los mensajes codificados sean canónicos, se definen las siguientes restricciones al codificar:

- Todos los registros dentro de un flujo TLV deben codificarse en orden de tipo estrictamente creciente.
- Todos los registros deben codificar como mínimo los campos de tipo y longitud. En otras palabras, se debe usar en todo momento la representación BigSize más pequeña para un entero.
- Cada tipo solo puede aparecer una vez dentro de un flujo TLV determinado.

Además de estas restricciones de codificación, también se define una serie de requisitos de interpretación de nivel superior en función de la ***aridad*** de un número entero de tipo dado. Nos sumergimos más en estos detalles hacia el final del capítulo una vez que describimos cómo se actualiza Lightning Protocol en la práctica y en la teoría.

Bits de función y extensibilidad de protocolo

Debido a que Lightning Network es un sistema descentralizado, ninguna entidad individual puede imponer un cambio o modificación de protocolo a todos los usuarios del sistema. Esta característica también se ve en otras redes descentralizadas como Bitcoin. Sin embargo, a diferencia de Bitcoin, ***no se*** requiere un consenso abrumador para cambiar un subconjunto de Lightning Network. Lightning puede evolucionar a voluntad sin un fuerte requisito de coordinación porque, a diferencia de Bitcoin, no se requiere un consenso global en Lightning Network.
Debido a este hecho y a los diversos mecanismos de actualización integrados en Lightning Network, solo los participantes que deseen utilizar estos nuevos

Las características de Lightning Network deben actualizarse y luego pueden interactuar entre sí.

En esta sección, exploramos las diversas formas en que los desarrolladores y usuarios pueden diseñar e implementar nuevas funciones en Lightning Network. Los diseñadores de Lightning Network original sabían que había muchas posibles direcciones futuras para la red y el protocolo subyacente. Como resultado, se aseguraron de implementar varios mecanismos de extensibilidad dentro del sistema, que pueden usarse para actualizarlo parcial o totalmente de manera desacoplada, desincronizada y descentralizada.

Feature Bits como un mecanismo de descubrimiento de actualizaciones

Un lector astuto puede haber notado las diversas ubicaciones donde se incluyen bits de características dentro del Protocolo Lightning. Un ***bit de función*** es un campo de bits que se puede utilizar para anunciar la comprensión o el cumplimiento de una posible actualización del protocolo de red. Los bits de función se asignan comúnmente en pares, lo que significa que cada nueva función/actualización potencial siempre define dos bits dentro del campo de bits. Un bit indica que la función anunciada es ***opcional,*** lo que significa que el nodo conoce la función y puede usarla, pero no la considera necesaria para el funcionamiento normal. El otro bit indica que se ***requiere la función,*** lo que significa que el nodo no continuará funcionando si un posible par no comprende esa función.

Usando estos dos bits (opcional y requerido), podemos construir una matriz de compatibilidad simple que los nodos/usuarios pueden consultar para determinar si un par es compatible con una función deseada, como se muestra en la Tabla 13-2.

y

metro

a

t

r

i

x

Tipo de bit	Remoto opcional	Requiere remoto	Remoto desconocido
Local opcional ÿ	ÿ	ÿ	
Se requiere local ÿ	ÿ	ÿ	
Local desconocido ÿ	ÿ	ÿ	

A partir de esta matriz de compatibilidad simplificada, podemos ver que mientras el otra parte conoce nuestro bit de característica, entonces podemos interactuar con ellos utilizando el protocolo. Si la fiesta ni siquiera sabe de qué somos refiriéndose ***y*** requieren la función, entonces somos incompatibles con a ellos. Dentro de la red, las características opcionales se señalan mediante un ***bit impar número,*** mientras que las funciones requeridas se señalan mediante un ***número de bit par.*** Como un ejemplo, si un compañero señala que conoce una característica que usa el bit 15, entonces sabemos que esta es una característica opcional, y podemos interactuar con ellos o responder a sus mensajes incluso si no conocemos la función. Si en su lugar, señalaron la característica usando el bit 16, entonces sabemos que esto es un característica requerida, y no podemos interactuar con ellos a menos que nuestro nodo también entiende esa característica.

Los desarrolladores de Lightning han ideado una frase fácil de recordar que codifica esta matriz: "está bien ser impar". Esta simple regla permite una rico conjunto de interacciones dentro del protocolo, como una simple operación de máscara de bits entre dos vectores de bits de características permite a los pares determinar si ciertos

las interacciones son compatibles entre sí o no. En otras palabras, los bits de funciones se utilizan como un mecanismo de detección de actualizaciones: permiten que los pares entiendan fácilmente si son compatibles o no en función de los conceptos de bits de funciones opcionales, requeridos y desconocidos.

Los bits de característica se encuentran en los mensajes node_announcement, channel_announcement e init dentro del protocolo. Como resultado, estos tres mensajes se pueden utilizar para señalar el conocimiento y/o la comprensión de las actualizaciones de protocolos en vuelo dentro de la red. Los bits de función que se encuentran en el mensaje node_announcement pueden permitir que un compañero determine si sus ***conexiones*** son compatibles o no. Los bits de característica dentro de los mensajes channel_announcement permiten que un par determine si un tipo de pago o HTLC determinado puede transitar a través de un par determinado o no. Los bits de función dentro del mensaje de inicio permiten a los pares comprender si pueden mantener una conexión y también qué funciones se negocian durante la vida útil de una conexión determinada.

TLV para compatibilidad hacia adelante y hacia atrás

Como aprendimos anteriormente en el capítulo, los registros TLV se pueden usar para extender mensajes de manera compatible hacia adelante y hacia atrás. Con el tiempo, estos registros se han utilizado para ampliar los mensajes existentes sin romper el protocolo al utilizar el área "indefinida" dentro de un mensaje más allá de ese conjunto de bytes conocidos.

Como ejemplo, el Lightning Protocol original no tenía un concepto de "mayor cantidad de HTLC" que podría atravesar un canal según lo dicta una política de enrutamiento. Posteriormente, se agregó el campo max_htlc al mensaje channel_update para incorporar este concepto a lo largo del tiempo. Los pares que reciben una actualización de canal que establece dicho campo pero que ni siquiera saben que la actualización existió no se ven afectados por el cambio, pero se rechazan sus HTLC si están más allá del límite. Los pares más nuevos, por otro lado, pueden analizar, verificar y utilizar el nuevo campo.

Quienes estén familiarizados con el concepto de bifurcaciones suaves en Bitcoin ahora pueden ver algunas similitudes entre los dos mecanismos. A diferencia del nivel de consenso de Bitcoin

bifurcaciones suaves, las actualizaciones a Lightning Network no requieren un consenso abrumador para ser adoptadas. En cambio, como mínimo, solo dos pares dentro de la red deben comprender una nueva actualización para comenzar a usarla. Por lo general, estos dos pares pueden ser el destinatario y el remitente de un pago, o pueden ser socios de canal de un nuevo canal de pago.

Una taxonomía de los mecanismos de actualización

En lugar de que exista un único mecanismo de actualización ampliamente utilizado dentro de la red (como bifurcaciones blandas para Bitcoin), existen varios mecanismos de actualización posibles dentro de Lightning Network. En esta sección, enumeramos estos mecanismos de actualización y brindamos un ejemplo real de su uso en el pasado.

Actualizaciones de red interna

Comenzamos con el tipo de actualización que requiere la mayor coordinación a nivel de protocolo: actualizaciones de red interna. Una actualización de red interna se caracteriza por una que requiere que ***cada nodo*** dentro de una posible ruta de pago comprenda la nueva función. Dicha actualización es similar a cualquier actualización dentro de Internet que requiera actualizaciones a nivel de hardware dentro de la parte de retransmisión central de la actualización. Sin embargo, en el contexto de Lightning Network, tratamos con software puro, por lo que tales actualizaciones son más fáciles de implementar, pero aún requieren mucha más coordinación que cualquier otro mecanismo de actualización en la red.

Un ejemplo de dicha actualización dentro de la red fue la introducción de una codificación TLV para la información de enrutamiento codificada dentro de los paquetes cebolla. El formato anterior usaba un formato de mensaje de longitud fija codificada para comunicar información como el siguiente salto. Debido a que este formato fue corregido, significaba que las nuevas actualizaciones de nivel de protocolo no eran posibles. El cambio al formato TLV más flexible significó que después de esta actualización, cualquier tipo de función que modificara el tipo de información comunicada en cada salto podría implementarse a voluntad.

Vale la pena mencionar que la actualización de la cebolla TLV fue una especie de actualización de red interna "suave", en el sentido de que si un pago no usaba ninguna característica nueva más allá de la nueva codificación de información de enrutamiento, entonces un pago podría transmitirse usando un conjunto mixto de nodos .

Actualizaciones de extremo a extremo

Para contrastar la actualización de la red interna, en esta sección describimos la actualización de la red ***de extremo a extremo .*** Este mecanismo de actualización difiere de la actualización de la red interna en que solo requiere que los "extremos" del pago, el remitente y el destinatario, se actualicen.

Este tipo de actualización permite una amplia gama de innovaciones sin restricciones dentro de la red. Debido a la naturaleza encriptada de cebolla de los pagos dentro de la red, es posible que aquellos que reenvían HTLC dentro del centro de la red ni siquiera sepan que se están utilizando nuevas funciones.

Un ejemplo de una actualización de extremo a extremo dentro de la red fue la implementación de pagos multiparte (MPP). MPP es una característica a nivel de protocolo que permite que un solo pago se divida en múltiples partes o rutas, para ensamblarse en el destinatario para su liquidación. El lanzamiento de MPP se combinó con un nuevo bit de función de nivel de anuncio de nodo que indica que el destinatario sabe cómo manejar pagos parciales. Suponiendo que un remitente y un destinatario se conocen (posiblemente a través de una factura BOLT #11), entonces pueden usar la nueva función sin más negociaciones.

Otro ejemplo de una actualización de extremo a extremo son los diversos tipos de pagos ***espontáneos*** implementados dentro de la red. Uno de los primeros tipos de pagos espontáneos llamado envío de ***claves*** funcionaba simplemente colocando la preimagen de un pago dentro de la cebolla cifrada. Una vez recibido, el destino descifraría la preimagen y luego la usaría para liquidar el pago.
Debido a que todo el paquete está encriptado de extremo a extremo, este tipo de pago era seguro, ya que ninguno de los nodos intermedios puede desenvolver completamente la cebolla para descubrir la preimagen del pago.

Actualizaciones a nivel de construcción del canal

La última categoría amplia de actualizaciones son aquellas que ocurren en el nivel de construcción del canal, pero que no modifican la estructura del HTLC que se usa ampliamente dentro de la red. Cuando decimos construcción del canal, nos referimos a cómo se financia o crea el canal. Como ejemplo, el tipo de canal de eltoo se puede implementar dentro de la red utilizando un nuevo bit de función de nivel de anuncio_nodo, así como un bit de función de nivel de anuncio_canal.
Solo los dos pares a los lados de los canales deben comprender y anunciar estas nuevas funciones. Este par de canales se puede usar para reenviar cualquier tipo de pago que el canal admita.

Otro es el formato de canal ***de salidas ancla*** que permite que la tarifa de compromiso se aumente a través del mecanismo de gestión de tarifas Child-Pays-For-Parent (CPFP) de Bitcoin.

Conclusión

El protocolo de cable de Lightning es increíblemente flexible y permite una rápida innovación e interoperabilidad sin un consenso estricto. Es una de las razones por las que Lightning Network está experimentando un desarrollo mucho más rápido y es atractivo para muchos desarrolladores, que de otro modo podrían encontrar el estilo de desarrollo de Bitcoin demasiado conservador y lento.

Capítulo 14. Transporte de mensajes cifrados de Lightning

En este capítulo, revisaremos el ***transporte de mensajes cifrados de Lightning Network,*** a veces denominado ***Protocolo Brontide,*** que permite a los pares establecer una comunicación cifrada, autenticación y verificación de integridad de extremo a extremo.

> **NOTA**
>
> Parte de este capítulo incluye algunos detalles muy técnicos sobre el protocolo de cifrado y los algoritmos de cifrado utilizados en el transporte cifrado Lightning. Puede decidir omitir esa sección si no está interesado en esos detalles.

Transporte encriptado en el relámpago

Conjunto de protocolos

El componente de transporte de Lightning Network y sus diversos componentes se muestran en la parte más a la izquierda de la capa de conexión de red en la Figura 14-1.

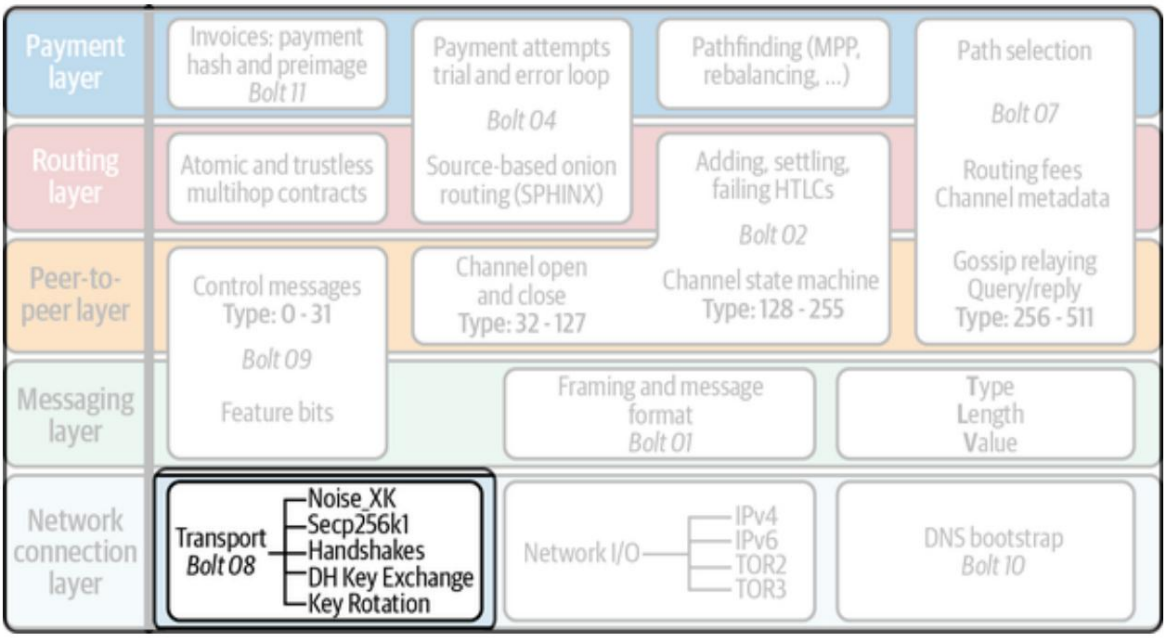

Figura 14-1. Transporte de mensajes cifrados en el conjunto de protocolos Lightning

Introducción

A diferencia de la red Vanilla Bitcoin P2P, cada nodo en Lightning Network se identifica mediante una clave pública única que sirve como su identidad. De forma predeterminada, esta clave pública se utiliza para cifrar de extremo a extremo ***todas*** las comunicaciones dentro de la red. El cifrado por defecto en el nivel más bajo del protocolo garantiza que todos los mensajes estén autenticados, sean inmunes a los ataques de intermediarios (MITM) y al espionaje de terceros, y garantiza la privacidad en el nivel de transporte fundamental. En este capítulo, aprenderemos en detalle sobre el protocolo de encriptación utilizado por Lightning Network. Al finalizar este capítulo, el lector estará familiarizado con el estado del arte de los protocolos de mensajería cifrada, así como con las diversas propiedades que dicho protocolo proporciona a la red. Vale la pena mencionar que el núcleo del transporte de mensajes cifrados es ***independiente*** de su uso dentro del contexto de Lightning Network. Como resultado, el transporte de mensajes cifrados personalizados que utiliza Lightning puede colocarse en cualquier contexto que requiera comunicación cifrada entre dos partes.

El gráfico de canales como público descentralizado Infraestructura clave

Como aprendimos en el Capítulo 8, cada nodo tiene una identidad a largo plazo que se usa como identificador de un vértice durante la búsqueda de rutas y también se usa en las operaciones criptográficas asimétricas relacionadas con la creación de paquetes de enrutamiento cifrados con cebolla. Esta clave pública, que sirve como identidad a largo plazo de un nodo, se incluye en la respuesta de arranque de DNS, así como también se integra en el gráfico de canal. Como resultado, antes de que un nodo intente conectarse a otro nodo en la red P2P, ya conoce la clave pública del nodo al que desea conectarse.

Además, si el nodo al que se conecta ya tiene una serie de canales públicos dentro del gráfico, entonces el nodo de conexión puede verificar aún más la identidad del nodo. Debido a que todo el gráfico de canales está completamente autenticado, uno puede verlo como una especie de infraestructura de clave pública (PKI) descentralizada: para registrar una clave, se debe abrir un canal público en la cadena de bloques de Bitcoin, y una vez que un nodo ya no tiene ningún canal público. canales, entonces se han eliminado efectivamente de la PKI.

Debido a que Lightning es una red descentralizada, es imperativo que ninguna parte central tenga el poder de proporcionar una identidad de clave pública dentro de la red. En lugar de una parte central, Lightning Network utiliza la cadena de bloques de Bitcoin como un mecanismo de mitigación de Sybil porque obtener una identidad en la red tiene un costo tangible: la tarifa necesaria para crear un canal en la cadena de bloques, así como el costo de oportunidad de la capital asignado a sus canales. En el proceso de implementar esencialmente una PKI específica de dominio, Lightning Network puede simplificar significativamente su protocolo de transporte encriptado, ya que no necesita lidiar con todas las complejidades que conlleva TLS, el protocolo de seguridad de la capa de transporte.

¿Por qué no TLS?

Los lectores familiarizados con el sistema TLS pueden preguntarse en este punto: ¿por qué no se usó TLS a pesar de las desventajas del sistema PKI existente? De hecho, es un hecho que los "certificados autofirmados" se pueden usar para eludir de manera efectiva el sistema PKI global existente simplemente afirmando la identidad de una clave pública determinada entre un conjunto de pares. Sin embargo, incluso con el sistema PKI existente fuera del camino, TLS tiene varios inconvenientes que llevaron a los creadores de Lightning Network a optar por un protocolo de cifrado personalizado más compacto.

Para empezar, TLS es un protocolo que existe desde hace varias décadas y, como resultado, ha evolucionado con el tiempo a medida que se han realizado nuevos avances en el espacio del cifrado de transporte. Sin embargo, con el tiempo, esta evolución ha provocado que el protocolo aumente en tamaño y complejidad. En las últimas décadas, se han descubierto y reparado varias vulnerabilidades en TLS, y cada evolución aumenta aún más la complejidad del protocolo. Como resultado de la antigüedad del protocolo, existen varias versiones e iteraciones, lo que significa que un cliente necesita comprender muchas de las iteraciones anteriores del protocolo para comunicarse con una gran parte de la Internet pública, lo que aumenta aún más la complejidad de la implementación.

En el pasado, se descubrieron varias vulnerabilidades de seguridad de la memoria en implementaciones ampliamente utilizadas de SSL/TLS. Empaquetar dicho protocolo dentro de cada nodo Lightning serviría para aumentar la superficie de ataque de los nodos expuestos a la red pública de igual a igual. Para aumentar la seguridad de la red en su conjunto y minimizar la superficie de ataque explotable, los creadores de Lightning Network optaron por adoptar Noise Protocol Framework. El ruido como protocolo internaliza varias de las lecciones de seguridad y privacidad aprendidas con el tiempo debido al escrutinio continuo del protocolo TLS durante décadas. En cierto modo, la existencia de Noise permite a la comunidad "comenzar de nuevo" de manera efectiva, con un protocolo más compacto y simplificado que conserva todos los beneficios adicionales de TLS.

El marco del protocolo de ruido

Noise Protocol Framework es un protocolo de cifrado de mensajes moderno, extensible y flexible diseñado por los creadores de Signal Protocol. El Signal Protocol es uno de los protocolos de cifrado de mensajes más utilizados en el mundo. Lo utilizan tanto Signal como Whatsapp, que acumuladamente son utilizados por más de mil millones de personas en todo el mundo. El marco Noise es el resultado de décadas de evolución tanto en el ámbito académico como en la industria de los protocolos de cifrado de mensajes. Lightning usa Noise Protocol Framework para implementar un protocolo de encriptación ***orientado*** a mensajes que todos los nodos usan para comunicarse entre sí.

Una sesión de comunicación que usa Noise tiene dos fases distintas: la fase de apretón de manos y la fase de mensajería. Antes de que dos partes puedan comunicarse entre sí, primero deben llegar a un secreto compartido que solo ellos conocen, que se utilizará para cifrar y autenticar los mensajes que se envían entre sí. Se utiliza un tipo de acuerdo de clave autenticado para llegar a una clave compartida final entre las dos partes. En el contexto del protocolo Noise, este acuerdo de clave autenticado se conoce como ***apretón de manos.*** Una vez que se ha completado ese apretón de manos, ambos nodos ahora pueden enviarse mensajes cifrados entre sí. Cada vez que los pares necesitan conectarse o reconectarse entre sí, se ejecuta una nueva iteración del protocolo de intercambio de señales, lo que garantiza que se logre el secreto hacia adelante (la filtración de la clave de una transcripción anterior no compromete ninguna transcripción futura).

Debido a que el Protocolo de ruido permite que un diseñador de protocolos elija entre varias primitivas criptográficas, como el cifrado simétrico y la criptografía de clave pública, es habitual que se haga referencia a cada versión del Protocolo de ruido con un nombre único. En el espíritu de "Ruido", cada sabor del protocolo selecciona un nombre derivado de algún tipo de "ruido". En el contexto de Lightning Network, el sabor del protocolo de ruido utilizado a veces se denomina Brontide. Un ***brontide*** es un ruido bajo y ondulante, similar a lo que uno escucharía durante una tormenta eléctrica cuando está muy lejos.

Transporte cifrado Lightning en detalle

En esta sección, desglosaremos el protocolo de transporte encriptado Lightning y profundizaremos en los detalles de los algoritmos y protocolos criptográficos utilizados para establecer comunicaciones encriptadas, autenticadas y con integridad asegurada entre pares. Siéntase libre de omitir esta sección si encuentra este nivel de detalle desalentador.

Noise_XK: Apretón de manos de ruido de Lightning Network

El protocolo de ruido es extremadamente flexible en el sentido de que anuncia varios protocolos de enlace, cada uno con diferentes propiedades de seguridad y privacidad para que un posible implementador del protocolo pueda seleccionar. Una exploración profunda de cada uno de los apretones de manos y sus diversas compensaciones está fuera del alcance de este capítulo. Dicho esto, Lightning Network utiliza un protocolo de enlace específico denominado Noise_XK. La propiedad única proporcionada por este protocolo de enlace es ***la ocultación de identidad:*** para que un nodo inicie una conexión con otro nodo, primero debe conocer su clave pública. Mecánicamente, esto significa que la clave pública del respondedor en realidad nunca se transmite durante el contexto del apretón de manos. En su lugar, se utiliza una serie inteligente de controles Elliptic Curve Diffie-Hellman (ECDH) y el código de autenticación de mensajes (MAC) para autenticar al respondedor.

Notación de protocolo de enlace y flujo de protocolo

Cada apretón de manos generalmente consta de varios pasos. En cada paso, se envía (posiblemente) material encriptado a la otra parte, se realiza un ECDH (o varios), con el resultado de que el apretón de manos se "mezcla" en una ***transcripción de protocolo.*** Esta transcripción sirve para autenticar cada paso del protocolo y ayuda a frustrar una especie de ataques de intermediarios. Al final del protocolo de enlace, se producen dos claves, ck y k, que se utilizan para cifrar mensajes (k) y rotar claves (ck) a lo largo de la duración de la sesión.

En el contexto de un apretón de manos, s suele ser una clave pública estática a largo plazo. En nuestro caso, el sistema criptográfico de clave pública utilizado es uno de curva elíptica, instanciado con la curva secp256k1, que se utiliza en otras partes de

Bitcoin. Se generan varias claves efímeras a lo largo del apretón de manos.
Usamos e para referirnos a una nueva clave efímera. Las operaciones ECDH entre dos claves se notan como la concatenación de dos claves. Como ejemplo, ee representa una operación ECDH entre dos claves efímeras.

Descripción general de alto nivel

Usando la notación presentada anteriormente, podemos describir sucintamente el Ruido_XK de la siguiente manera:

```
Ruido_XK(s, rs):
    <- rs
     ...
    -> e, e(rs) <- e,
    ee
    -> con, con
```

El protocolo comienza con la "pretransmisión" de la clave estática (rs) del respondedor al iniciador. Antes de ejecutar el protocolo de enlace, el iniciador debe generar su(s) propia(s) clave(s) estática(s). Durante cada paso del protocolo de enlace, todo el material enviado a través del cable y las claves enviadas/utilizadas se procesan gradualmente en un ***resumen del protocolo de enlace,*** h. Este resumen nunca se envía por cable durante el protocolo de enlace y, en cambio, se usa como "datos asociados" cuando se envía AEAD (cifrado autenticado con datos asociados) por cable. ***Los datos asociados*** (AD) permiten que un protocolo de cifrado autentique información adicional junto con un paquete de texto cifrado. En otros dominios, el AD puede ser un nombre de dominio o una porción de texto sin formato del paquete.

La existencia de h garantiza que si se reemplaza una parte de un mensaje de protocolo de enlace transmitido, el otro lado lo notará. En cada paso, se verifica un resumen de MAC. Si la verificación de MAC tiene éxito, entonces la parte receptora sabe que el protocolo de enlace ha sido exitoso hasta ese momento.
De lo contrario, si alguna vez falla una verificación de MAC, entonces el proceso de negociación ha fallado y la conexión debe terminarse.

El protocolo también agrega un nuevo dato a cada mensaje de protocolo de enlace: una versión del protocolo. La versión inicial del protocolo es 0. Al momento de escribir, no

se han creado nuevas versiones del protocolo. Como resultado, si un compañero recibe una versión distinta de 0, entonces debe rechazar el intento de inicio del protocolo de enlace.

En cuanto a las primitivas criptográficas, SHA-256 se utiliza como la función hash de elección, secp256k1 como la curva elíptica y ChaChaPoly-130 como la construcción AEAD (cifrado simétrico).

Cada variante del protocolo de ruido tiene una cadena ASCII única que se utiliza para referirse a ella. Para garantizar que las dos partes utilicen la misma variante de protocolo, la cadena ASCII se convierte en un resumen, que se utiliza para inicializar el estado inicial del protocolo de enlace. En el contexto de Lightning Network, la cadena ASCII que describe el protocolo es Noise_XK_secp256k1_ChaChaPoly_SHA256.

Apretón de manos en tres actos

La parte del apretón de manos se puede separar en tres "actos" distintos. El apretón de manos completo toma 1,5 viajes de ida y vuelta entre el iniciador y el respondedor. En cada acto, se envía un único mensaje entre ambas partes. El mensaje de protocolo de enlace es una carga útil de tamaño fijo precedido por la versión del protocolo.

El protocolo de ruido utiliza una notación inspirada en objetos para describir el protocolo en cada paso. Durante la configuración del estado de protocolo de enlace, cada lado inicializará las siguientes variables:

ck

> La ***llave del encadenamiento.*** Este valor es el hash acumulado de todas las salidas ECDH anteriores. Al final del protocolo de enlace, se usa ck para derivar las claves de cifrado para los mensajes Lightning.

h

> El ***hash del apretón de manos.*** Este valor es el hash acumulado de ***todos los*** datos de protocolo de enlace que se han enviado y recibido hasta el momento durante el proceso de protocolo de enlace.

temp_k1, temp_k2, temp_k3

Las ***claves intermedias.*** Estos se utilizan para cifrar y descifrar las cargas útiles de AEAD de longitud cero al final de cada mensaje de protocolo de enlace.

y

El ***efímero par de llaves de una fiesta.*** Para cada sesión, un nodo debe generar una nueva clave efímera con fuerte aleatoriedad criptográfica.

s

El par de ***llaves estáticas*** de una parte (ls para local, rs para remoto).

Dado este estado de sesión de protocolo de enlace más mensajería, definiremos una serie de funciones que operarán en el estado de protocolo de enlace y mensajería. Al describir el protocolo de protocolo de enlace, usaremos estas variables de manera similar al pseudocódigo para reducir la verbosidad de la explicación de cada paso del protocolo. Definiremos las primitivas ***funcionales*** del apretón de manos como:

ECDH(k, rk)

Realiza una operación Diffie-Hellman de curva elíptica utilizando k, que es una clave privada secp256k1 válida, y rk, que es una clave pública válida.

El valor devuelto es el SHA-256 del formato comprimido del punto generado.

HKDF (sal, ikm)

Una función definida en RFC 5869, evaluada con un campo de información de longitud cero.

Todas las invocaciones de HKDF devuelven implícitamente 64 bytes de aleatoriedad criptográfica utilizando el componente de extracción y expansión de HKDF.

cifrar con anuncio (k, n, anuncio, texto sin formato)

Las salidas cifran (k, n, ad, texto sin formato).

Donde encrypt es una evaluación de ChaCha20-Poly1305 (variante del Grupo de trabajo de ingeniería de Internet) con los argumentos pasados, con nonce n codificado como 32 bits cero, seguido de un valor ***little-endian*** de 64 bits.
Nota: esto sigue la convención del protocolo de ruido, en lugar de nuestro endian normal.

decryptWithAD(k, n, anuncio, texto cifrado)

Salidas descifradas (k, n, ad, texto cifrado).

Donde descifrar es una evaluación de ChaCha20-Poly1305 (variante IETF) con los argumentos pasados, con nonce n codificado como 32 bits cero, seguido de un valor ***little-endian*** de 64 bits.

generar clave ()

Genera y devuelve un nuevo par de claves secp256k1.

Donde el objeto devuelto por generateKey tiene dos atributos: .pub, que devuelve un objeto abstracto que representa la clave pública; y .priv, que representa la clave privada utilizada para generar la clave pública

Donde el objeto también tiene un solo método: .serializeCompressed()

un || *b*

Esto denota la concatenación de dos cadenas de bytes a y b.

Inicialización del estado de la sesión de protocolo de enlace

Antes de iniciar el proceso de protocolo de enlace, ambas partes deben inicializar el estado inicial que usarán para avanzar en el proceso de protocolo de enlace. Para comenzar, ambas partes deben construir el resumen inicial del apretón de manos h.

1. h = *SHA-256 (nombre de protocolo)*

Donde *protocolName* = "Noise_XK_secp256k1_ChaChaPoly_SHA256" codificado como una cadena ASCII.

2. ck = h

3. h = SHA-256(h || *prólogo)*

   Donde *prólogo* es la cadena ASCII: relámpago.

Además del nombre del protocolo, también agregamos un "prólogo" adicional que se usa para vincular aún más el contexto del protocolo a Lightning Network.

Para concluir el paso de inicialización, ambos lados mezclan la clave pública del respondedor en el resumen del protocolo de enlace. Debido a que este resumen se usa mientras se envían los datos asociados con un texto cifrado de longitud cero (solo el MAC), esto garantiza que el iniciador realmente conozca la clave pública del respondedor.

- El nodo iniciador mezcla la clave pública estática del nodo que responde serializada en el formato comprimido de Bitcoin: h = SHA-256(h || rs.pub.serializeCompressed())
- El nodo que responde se mezcla en su clave pública estática local serializada en el formato comprimido de Bitcoin: h = SHA-256(h || ls.pub.serializeCompressed())

actos de apretón de manos

Después de la inicialización del protocolo de enlace inicial, podemos comenzar la ejecución real del proceso de protocolo de enlace. El apretón de manos se compone de una serie de tres mensajes enviados entre el iniciador y el respondedor, en lo sucesivo denominados "actos". Debido a que cada acto es un mensaje único enviado entre las partes, un apretón de manos se completa en un total de 1,5 viajes de ida y vuelta (0,5 por cada acto).

El primer acto completa la parte inicial del intercambio de claves incremental triple Diffie-Hellman (DH) (usando una nueva clave efímera generada por el iniciador) y también asegura que el iniciador realmente conozca la clave pública a largo plazo del respondedor. Durante el segundo acto, el respondedor transmite la clave efímera que desea utilizar para la sesión al iniciador, y una vez

nuevamente mezcla de forma incremental esta nueva clave en el protocolo de enlace triple DH. Durante el tercer y último acto, el iniciador transmite su clave pública estática a largo plazo al respondedor y ejecuta la operación DH final para mezclarla en el secreto compartido resultante final.

acto uno

-> e, es

El primer acto se envía del iniciador al respondedor. Durante el primer acto, el iniciador intenta satisfacer un desafío implícito del respondedor. Para completar este desafío, el iniciador debe conocer la clave pública estática del respondedor.

El mensaje de reconocimiento tiene ***exactamente*** 50 bytes: 1 byte para la versión de reconocimiento, 33 bytes para la clave pública efímera comprimida del iniciador y 16 bytes para la etiqueta poly1305.

Acciones del remitente:

1. e = generar clave()
2. h = SHA-256(h || e.pub.serializeCompressed())

   La clave efímera recién generada se acumula en el resumen de protocolo de enlace en ejecución.

3. es = ECDH(e.priv, rs)

   El iniciador realiza un ECDH entre su clave efímera recién generada y la clave pública estática del nodo remoto.

4. ck, temp_k1 = HKDF(ck, es)

   Se genera una nueva clave de cifrado temporal, que se utiliza para generar la MAC de autenticación.

5. c = cifrar con anuncio (temp_k1, 0, h, cero)

   Donde cero es un texto sin formato de longitud cero.

6. h = SHA-256(h || c)

Finalmente, el texto cifrado generado se acumula en el resumen del protocolo de enlace de autenticación.

7. Enviar m = 0 || e.pub.serializeCompressed() || C
   al respondedor a través del búfer de la red.

Acciones del receptor:

1. Lea ***exactamente*** 50 bytes del búfer de red.
2. Analice el mensaje leído (m) en v, re y c:
   - Donde v es el ***primer*** byte de m, re son los siguientes 33 bytes de m y c son los últimos 16 bytes de m.
   - Los bytes sin procesar de la clave pública efímera (re) de la parte remota deben deserializarse en un punto de la curva utilizando coordenadas afines codificadas por el formato compuesto serializado de la clave.
3. Si v es una versión de protocolo de enlace no reconocida, el respondedor debe cancelar el intento de conexión.
4. h = SHA-256(h || re.serializeCompressed())

   El respondedor acumula la clave efímera del iniciador en el resumen del protocolo de enlace de autenticación.
5. es = ECDH(s.priv, re)

   El respondedor realiza un ECDH entre su clave privada estática y la clave pública efímera del iniciador.
6. ck, temp_k1 = HKDF(ck, es)

   Se genera una nueva clave de cifrado temporal, que se utilizará en breve para verificar la MAC de autenticación.
7. p = descifrar con AD (temp_k1, 0, h, c)

Si falla la comprobación de MAC en esta operación, el iniciador ***no*** conoce la clave pública estática del respondedor. Si este es el caso, entonces el respondedor debe terminar la conexión sin más
mensajes

8. h = SHA-256(h || c)

   El texto cifrado recibido se mezcla en el resumen del protocolo de enlace. Este paso sirve para garantizar que un MITM no haya modificado la carga útil.

segundo acto

<- e, sí

El segundo acto se envía desde el respondedor al iniciador. El segundo acto ***solo*** tendrá lugar si el primer acto tuvo éxito. El primer acto tuvo éxito si el respondedor pudo descifrar correctamente y verificar el MAC de la etiqueta enviada al final del primer acto.

El protocolo de enlace tiene ***exactamente*** 50 bytes: 1 byte para la versión de protocolo de enlace, 33 bytes para la clave pública efímera comprimida del respondedor y 16 bytes para la etiqueta poly1305.

Acciones del remitente:

1. e = generar clave()
2. h = SHA-256(h || e.pub.serializeCompressed())

   La clave efímera recién generada se acumula en el resumen de protocolo de enlace en ejecución.

3. ee = ECDH(e.priv, re)

   Donde re es la llave efímera del iniciador, que fue recibida durante el Primer Acto.

4. ck, temp_k2 = HKDF(ck, ee)

Se genera una nueva clave de cifrado temporal, que se utiliza para generar la MAC de autenticación.

5. c = cifrar con AD (temp_k2, 0, h, cero)

   Donde cero es un texto sin formato de longitud cero.

6. h = SHA-256(h || c)

   Finalmente, el texto cifrado generado se acumula en el resumen del protocolo de enlace de autenticación.

7. Enviar m = 0 || e.pub.serializeCompressed() || C al iniciador a través del búfer de la red.

Acciones del receptor:

1. Lea ***exactamente*** 50 bytes del búfer de red.
2. Analice el mensaje leído (m) en v, re y c:

   Donde v es el ***primer*** byte de m, re son los siguientes 33 bytes de m y c son los últimos 16 bytes de m.

3. Si v es una versión de protocolo de enlace no reconocida, el respondedor debe cancelar el intento de conexión.
4. h = SHA-256(h || re.serializeCompressed())
5. ee = ECDH(e.priv, re)

   Donde re es la clave pública efímera del respondedor.

   Los bytes sin procesar de la clave pública efímera (re) de la parte remota deben deserializarse en un punto de la curva utilizando coordenadas afines codificadas por el formato compuesto serializado de la clave.

6. ck, temp_k2 = HKDF(ck, ee)

   Se genera una nueva clave de cifrado temporal, que se utiliza para generar la MAC de autenticación.

7. p = descifrar con AD (temp_k2, 0, h, c)

   Si la verificación de MAC en esta operación falla, entonces el iniciador debe terminar la conexión sin más mensajes.

8. h = SHA-256(h || c)

   El texto cifrado recibido se mezcla en el resumen del protocolo de enlace. Este paso sirve para garantizar que un MITM no haya modificado la carga útil.

tercer acto

-> con, con

El tercer acto es la fase final del acuerdo de clave autenticada descrito en esta sección. Este acto se envía desde el iniciador al respondedor como un paso final. El tercer acto se ejecuta ***si y solo si*** el segundo acto tuvo éxito. Durante el tercer acto, el iniciador transporta su clave pública estática al respondedor cifrada con ***un fuerte*** secreto hacia adelante, utilizando la clave secreta acumulada derivada de HKDF en este punto del protocolo de enlace.

El protocolo de enlace tiene ***exactamente*** 66 bytes: 1 byte para la versión de protocolo de enlace, 33 bytes para la clave pública estática cifrada con el cifrado de flujo ChaCha20, 16 bytes para la etiqueta de la clave pública cifrada generada a través de la construcción AEAD y 16 bytes para una etiqueta de autenticación final .

Acciones del remitente:

1. c = encryptWithAD(temp_k2, 1, h, s.pub.serializeCompressed())

   Donde s es la clave pública estática del iniciador.

2. h = SHA-256(h || c)

3. se = ECDH(s.priv, re)

   Donde re es la clave pública efímera del respondedor.

4. ck, temp_k3 = HKDF(ck, se)

El secreto compartido intermedio final se mezcla con la clave de encadenamiento en ejecución.

5. t = cifrar con AD (temp_k3, 0, h, cero)

   Donde cero es un texto sin formato de longitud cero.

6. sk, rk = HKDF(ck, cero)

   Donde cero es un texto sin formato de longitud cero, sk es la clave que utilizará el iniciador para cifrar los mensajes al respondedor, y rk es la clave que utilizará el iniciador para descifrar los mensajes enviados por el respondedor.

   Se generan las claves de cifrado finales, que se utilizarán para enviar y recibir mensajes durante la duración de la sesión.

7. rn = 0, sn = 0

   Los nonces de envío y recepción se inicializan a 0.

8. Enviar m = 0 || do || t sobre el búfer de la red.

Acciones del receptor:

1. Lea ***exactamente*** 66 bytes del búfer de red.
2. Analice el mensaje leído (m) en v, c y t:

   Donde v es el ***primer*** byte de m, c son los siguientes 49 bytes de m y t son los últimos 16 bytes de m.

3. Si v es una versión de protocolo de enlace no reconocida, el respondedor debe cancelar el intento de conexión.
4. rs = descifrar con AD (temp_k2, 1, h, c)

   En este punto, el respondedor ha recuperado la clave pública estática del iniciador.

5. h = SHA-256(h || c)

6. se = ECDH(e.priv, rs)

   Donde e es la clave efímera original del respondedor.

7. ck, temp_k3 = HKDF(ck, se)

8. p = descifrar con AD (temp_k3, 0, h, t)

   Si la comprobación de MAC en esta operación falla, el respondedor debe terminar la conexión sin más mensajes.

9. rk, sk = HKDF(ck, cero)

   Donde cero es un texto sin formato de longitud cero, rk es la clave que utilizará el respondedor para descifrar los mensajes enviados por el iniciador, y sk es la clave que utilizará el respondedor para cifrar los mensajes al iniciador.

   Se generan las claves de cifrado finales, que se utilizarán para enviar y recibir mensajes durante la duración de la sesión.

10. rn = 0, sn = 0

    Los nonces de envío y recepción se inicializan a 0.

Cifrado de mensajes de transporte Al

finalizar el tercer acto, ambas partes han derivado las claves de cifrado, que se utilizarán para cifrar y descifrar mensajes durante el resto de la sesión.

Los mensajes reales del Protocolo Lightning están encapsulados dentro de los textos cifrados de AEAD. Cada mensaje tiene el prefijo de otro texto cifrado AEAD, que codifica la longitud total del siguiente mensaje Lightning (sin incluir su MAC).

El tamaño ***máximo*** de ***cualquier*** mensaje Lightning no debe exceder los 65 535 bytes. Un tamaño máximo de 65ÿ535 simplifica las pruebas, facilita la administración de la memoria y ayuda a mitigar los ataques de agotamiento de la memoria.

Para dificultar el análisis del tráfico, el prefijo de longitud de todos los mensajes Lightning encriptados también está encriptado. Además, se agrega una etiqueta Poly-1305 de 16 bytes al prefijo de longitud cifrada para garantizar que la longitud del paquete no se haya modificado durante el vuelo y también para evitar la creación de un oráculo de descifrado.

La estructura de los paquetes en el cable se parece al diagrama de la figura 14-2.

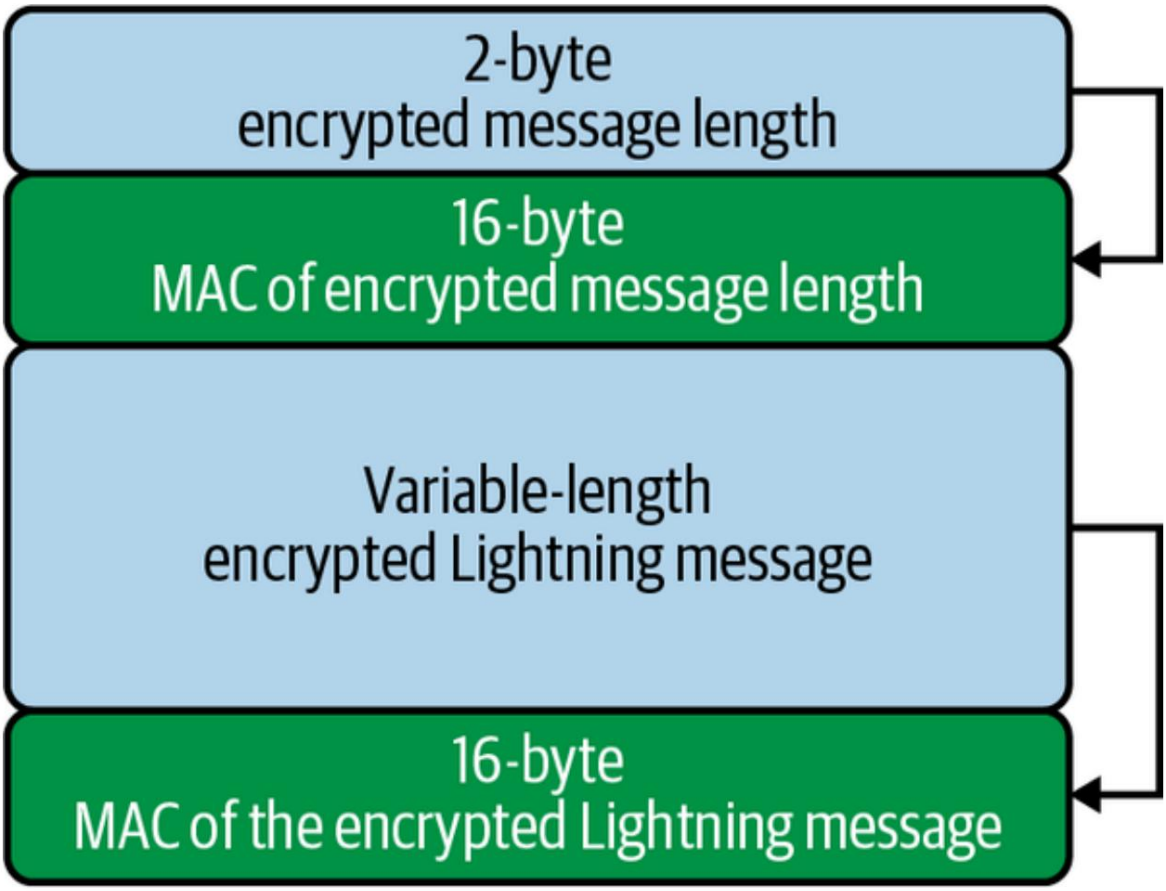

Figura 14-2. Estructura de paquetes encriptados

La longitud del mensaje con prefijo se codifica como un entero big-endian de 2 bytes, para una longitud máxima total del paquete de 2 + 16 + 65 535 + 16 = 65 569 bytes.

Cifrado y envío de mensajes

Para cifrar y enviar un mensaje Lightning (m) al flujo de red, dada una clave de envío (sk) y un nonce (sn), se completan los siguientes pasos:

1. Sea l = len(m).

Donde len obtiene la longitud en bytes del mensaje Lightning.

2. Serialice l en 2 bytes codificados como un entero big-endian.

3. Cifre l (usando ChaChaPoly-1305, sn y sk), para obtener lc (18 bytes).

    - El nonce sn está codificado como un número little-endian de 96 bits. Como el nonce decodificado es de 64 bits, el nonce de 96 bits se codifica como 32 bits de ceros iniciales seguidos de un valor de 64 bits.
    - El nonce sn debe incrementarse después de este paso.
    - Un segmento de bytes de longitud cero debe pasarse como AD (datos asociados).

4. Finalmente, cifre el mensaje en sí (m) utilizando el mismo procedimiento utilizado para cifrar el prefijo de longitud. Que este texto cifrado encriptado se conozca como c.

    El nonce sn debe incrementarse después de este paso.

5. Enviar lc || c sobre el búfer de la red.

Recibir y descifrar mensajes

Para descifrar el ***siguiente*** mensaje en el flujo de red, se completan los siguientes pasos:

1. Lea ***exactamente*** 18 bytes del búfer de red.

2. Deje que el prefijo de longitud cifrada se conozca como lc.

3. Descifre lc (usando ChaCha20-Poly1305, rn y rk) para obtener el tamaño del paquete cifrado l.

    - Un segmento de bytes de longitud cero debe pasarse como AD (datos asociados).
    - El nonce rn debe incrementarse después de este paso.

4. Lea ***exactamente*** l + 16 bytes del búfer de red y deje que el los bytes se conocen como c.
5. Descifre c (usando ChaCha20-Poly1305, rn y rk) para obtener el paquete de texto sin formato descifrado p.

   El nonce rn debe incrementarse después de este paso.

Rotación de claves de mensajes relámpago

El cambio de claves con regularidad y el olvido de claves anteriores es útil para evitar el descifrado de mensajes antiguos, en el caso de una fuga de claves posterior (es decir, secreto hacia atrás).

La rotación de claves se realiza para ***cada*** clave (sk y rk) ***individualmente.*** Una clave debe rotarse después de que una de las partes cifra o descifra 1000 veces con ella (es decir, cada 500 mensajes). Esto se puede contabilizar correctamente rotando la clave una vez que el nonce dedicado a ella supere los 1000.

La rotación de claves para una clave k se realiza de acuerdo con los siguientes pasos:

1. Sea ck la clave de encadenamiento obtenida al final del tercer acto.
2. ck', k' = HKDF(ck, k)
3. Restablezca el nonce para la clave en n = 0.
4. k = k'
5. ck = ck'

Conclusión

El cifrado de transporte subyacente de Lightning se basa en el protocolo de ruido y ofrece sólidas garantías de seguridad de privacidad, autenticidad e integridad para todas las comunicaciones entre pares de Lightning.

A diferencia de Bitcoin, donde los pares a menudo se comunican "en claro" (sin encriptación), todas las comunicaciones Lightning están encriptadas entre pares. En

Además del cifrado de transporte (peer-to-peer), en Lightning Network, los pagos ***también*** se cifran en paquetes de cebolla (hop-to-hop) y los detalles de pago se envían fuera de banda entre el remitente y el destinatario (end-to). -final). La combinación de todos estos mecanismos de seguridad es acumulativa y proporciona una defensa en capas contra la anonimización, los ataques de intermediarios y la vigilancia de la red.

Por supuesto, ninguna seguridad es perfecta y veremos en el Capítulo 16 que estas propiedades pueden degradarse y atacarse. Sin embargo, Lightning Network mejora significativamente la privacidad de Bitcoin.

Capítulo 15. Solicitudes de Pago Lightning

En este capítulo, veremos ***las solicitudes de pago Lightning,*** o como se las conoce más comúnmente, las ***facturas Lightning.***

Facturas en Lightning Protocol Suite

Las solicitudes de pago, también conocidas como ***facturas,*** son parte de la capa de pago y se muestran en la parte superior izquierda de la Figura 15-1.

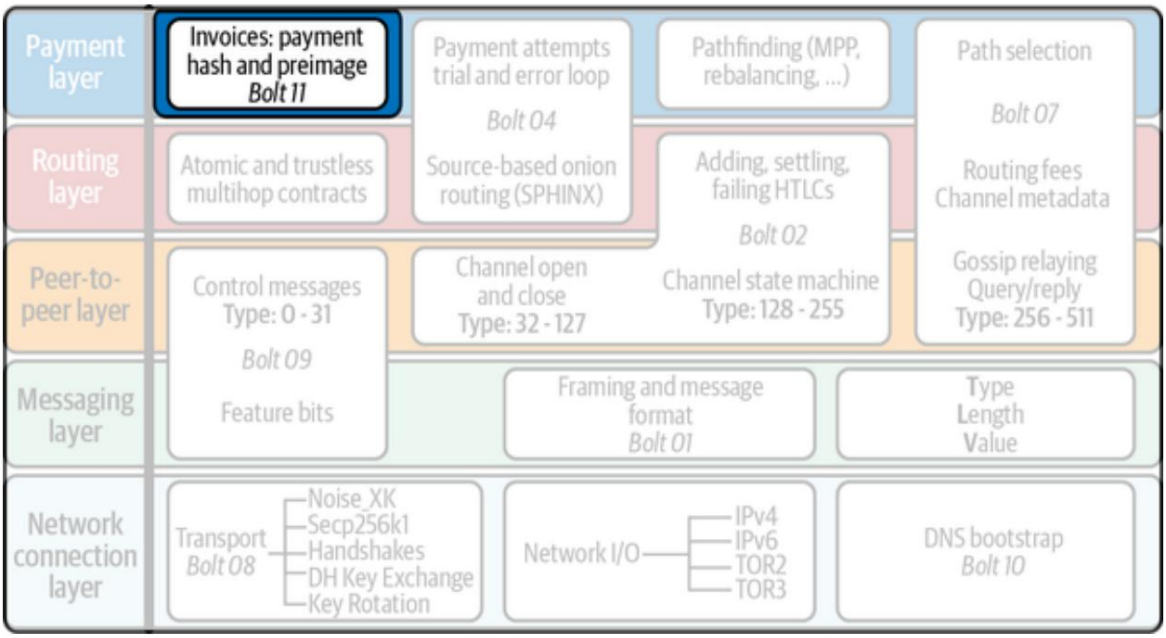

Figura 15-1. Solicitudes de pago en el conjunto de protocolos Lightning

Introducción

Como hemos aprendido a lo largo del libro, se requieren como mínimo dos datos para completar un pago Lightning: un hash de pago y un destino. Como SHA-256 se usa en Lightning Network para implementar

HTLC, esta información requiere 32 bytes para comunicarse. Los destinos, por otro lado, son simplemente la clave pública secp256k1 del nodo que desea recibir un pago. El propósito de una solicitud de pago en el contexto de Lightning Network es comunicar estos dos datos del remitente al destinatario. El formato amigable con el código QR para comunicar la información requerida para completar un pago del receptor al remitente se describe en el BOLT #11: Protocolo de factura para pagos Lightning. En la práctica, en una solicitud de pago se comunica algo más que el hash y el destino del pago para que la codificación sea más completa.

Solicitudes de pago relámpago frente a Bitcoin direcciones

Una pregunta frecuente cuando las personas se encuentran por primera vez con una solicitud de pago Lightning es: ¿por qué no se puede usar un formato de dirección estática normal en su lugar?

Para responder a esta pregunta, primero debe interiorizar en qué se diferencia Lightning de la capa base Bitcoin como método de pago. En comparación con una dirección de Bitcoin que puede usarse para realizar una cantidad potencialmente ilimitada de pagos (aunque reutilizar una dirección de Bitcoin puede degradar la privacidad), una solicitud de pago Lightning solo debe usarse ***una vez.*** Esto se debe al hecho de que enviar un pago a una dirección de Bitcoin utiliza esencialmente un sistema criptográfico de clave pública para "codificar" el pago de manera que solo el verdadero "propietario" de esa dirección de Bitcoin pueda canjearlo.

Por el contrario, para completar un pago Lightning, el destinatario debe revelar un "secreto" a toda la ruta de pago, incluido el remitente. Esto puede interpretarse como el uso de un tipo de criptografía simétrica específica del dominio porque la preimagen de pago es, a efectos prácticos, un nonce (el número solo se usa una vez). Si el remitente intenta realizar otro pago utilizando ese hash de pago idéntico, corre el riesgo de perder fondos porque es posible que el pago no se entregue en el destino. Es seguro asumir que después de que se haya revelado una preimagen, todos los nodos en la ruta la mantendrán.

para siempre, luego, en lugar de reenviar el HTLC para cobrar una tarifa de enrutamiento si se completa el pago, simplemente pueden liquidar el pago en esa instancia y obtener el monto total del pago a cambio. Como resultado, no es seguro usar una solicitud de pago más de una vez.

Existen nuevas variantes de la solicitud de pago Lightning original que permiten al remitente reutilizarlas tantas veces como quiera. Estas variantes invierten el flujo de pago normal ya que el remitente transmite una preimagen dentro de la carga útil de cebolla cifrada al receptor, que es el único que puede descifrarlo y liquidar el pago. Alternativamente, suponiendo un mecanismo que permita que un remitente solicite típicamente una nueva solicitud de pago del receptor, entonces se puede usar un protocolo interactivo para permitir un grado de reutilización de la solicitud de pago.

TORNILLO #11: Solicitud de pago de relámpago Serialización e Interpretación

En esta sección, describiremos el mecanismo utilizado para codificar el conjunto de información requerida para completar un pago en Lightning Network. Como se mencionó anteriormente, el hash y el destino del pago es la cantidad mínima de información requerida para completar un pago. Sin embargo, en la práctica, también se comunica más información, como información de bloqueo de tiempo, vencimiento de la solicitud de pago y posiblemente una dirección alternativa en la cadena. El documento de especificaciones completo es BOLT #11: Protocolo de factura para pagos relámpago.

Codificación de solicitud de pago en la práctica

Primero, examinemos cómo se ve una solicitud de pago real en la práctica. La siguiente es una solicitud de pago válida que podría haberse utilizado para completar un pago en la red principal Lightning Network en el momento en que se creó:

```
lnbc2500u1pvjluezpp5qqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqqqsyqcyq5rqw zqfqypqdq5xysx
xatsyp3k7enxv4jsxqzpuaztrnwngzn3kdzw5hydlzf03qdgm2hdq27cqv3
```

hz5se903vruatf
hq77w3ls4evs3ch9zw97j25emudupq63nyw24cg27h2rspfj9srp

El prefijo legible por humanos

Mirando la cadena, podemos extraer una parte que podemos analizar con nuestros ojos, mientras que el resto parece un conjunto aleatorio de cadenas. La parte que es algo analizable por un humano se denomina ***prefijo legible por humanos.*** Permite que un humano extraiga rápidamente información relevante de una solicitud de pago de un vistazo. En este caso, podemos ver que este pago es para la instancia de red principal de Lightning Network (lnbc), y está solicitando 2.500 uBTC (microbitcoin), o 25.000.000 satoshis. La última poción se conoce como la porción de datos y utiliza un formato extensible para codificar la información requerida para completar un pago.

Cada versión de instancia de Lightning Network (red principal, red de prueba, etc.) tiene su propio prefijo legible por humanos (consulte la Tabla 15-1). Esto permite que el software del cliente y también los humanos determinen rápidamente si su nodo puede satisfacer una solicitud de pago o no.

La red	Prefijo de PERNO #11
red principal	lnbc
red de prueba	Intb
simnet/registro	lnbcrt

La primera parte del prefijo legible por humanos es una expresión ***compacta*** de el monto de la solicitud de pago. La cantidad compacta está codificada en dos partes. En primer lugar, se utiliza un número entero como cantidad ***base .*** A continuación, esto es seguido por un multiplicador que nos permite especificar distintos aumentos de orden de magnitud compensado por el monto base. Si volvemos a nuestro ejemplo inicial, entonces podemos tome la porción de 2500u y disminúyala por un factor de 1,000 para usar en su lugar 2500m o (2500 mBTC). Como regla general, para determinar la cantidad de un factura de un vistazo, toma el factor base y multiplícalo por el multiplicador.

En la Tabla 15-2 se proporciona una lista completa de los multiplicadores actualmente definidos .

r

s

Multiplicador	unidad bitcoin	Factor de multiplicación
metro	nacional	0.001
en	micro	0.000001
norte	nano	0.000000001
pags	pico	0.000000000001

bech32 y el segmento de datos

Si la parte "ilegible" le resulta familiar, es porque utiliza el
el mismo esquema de codificación que usan las direcciones de Bitcoin compatibles con SegWit
hoy, a saber, bech32. Describir el esquema de codificación bech32 está fuera
el alcance de este capítulo. En resumen, es una forma sofisticada de codificar
cadenas que tiene muy buena corrección de errores, así como propiedades de detección.

La porción de datos se puede separar en tres secciones:

- la marca de tiempo
- Cero o más pares clave-valor etiquetados
- La firma de la factura completa.

La marca de tiempo se expresa en segundos desde el año 1970, o el Unix
Época. Esta marca de tiempo permite al remitente medir la antigüedad de la factura,
y como veremos más adelante, permite que el receptor fuerce una factura para que solo sea
válido por un período de tiempo si así lo desean.

Similar al formato TLV que aprendimos en "Tipo-Longitud-Valor
Format", el formato de factura BOLT #11 utiliza una serie de valores-clave extensibles
pares para codificar la información necesaria para satisfacer un pago. Porque clave-valor

se utilizan pares, es fácil agregar nuevos valores en el futuro si se introduce un nuevo tipo de pago o un requisito/funcionalidad adicional.

Finalmente, se incluye una firma que cubre toda la factura firmada por el destino del pago. Esta firma le permite al remitente verificar que la solicitud de pago fue efectivamente creada por el destino del pago.
A diferencia de las solicitudes de pago de Bitcoin que no están firmadas, esto nos permite asegurarnos de que una entidad en particular firmó la solicitud de pago. La firma en sí está codificada usando una identificación de recuperación, lo que permite usar una firma más compacta que permite la extracción de clave pública. Al verificar la firma, el ID de recuperación extrae la clave pública y luego la compara con la clave pública incluida en la factura.

Campos de factura etiquetados

Los campos de factura etiquetados están codificados en el cuerpo principal de la factura.
Estos campos representan diferentes pares clave-valor que expresan información adicional que puede ayudar a completar el pago o información que se ***requiere*** para completar el pago. Debido a que se utiliza una ligera variante de bech32, cada uno de estos campos está en realidad en el dominio "base 5".

Un campo de etiqueta determinado se compone de tres componentes:

- El tipo del campo (5 bits)
- La longitud de los datos del campo (10 bits)
- Los datos en sí, que tienen una longitud de * 5 bytes de tamaño

En la Tabla 15-3 se proporciona una lista completa de todos los campos etiquetados definidos actualmente .

yo

d

s

Etiqueta de campo	Longitud de datos	Uso
pags	52	El hash de pago SHA-256.
s	52	Un secreto de 256 bits que aumenta la privacidad de extremo a extremo de un pago mitigando el sondeo por nodos intermedios.
d	Variable	La descripción, una cadena corta UTF-8 del propósito de el pago.
norte	53	La clave pública del nodo de destino.
h	52	Un hash que representa una descripción del pago. sí mismo. Esto se puede usar para comprometerse con una descripción que es más de 639 bytes de longitud.
x	Variable	El tiempo de vencimiento, en segundos, del pago. El valor por defecto es 1 hora (3600) si no se especifica.
c	Variable	El min_cltv_expiry a usar para el salto final en el ruta. El valor predeterminado es 9 si no se especifica.
F	Variable	Una dirección alternativa en la cadena que se usará para completar el pago si el pago no se puede completar durante el Red Rayo.
r	Variable	Una o más entradas que permiten a un receptor dar la remitente bordes efímeros adicionales para completar el pago.
9	Variable	Un conjunto de valores de 5 bits que contienen los bits de característica que son necesarios para completar el pago.

Los elementos contenidos en el campo r se denominan comúnmente ***enrutamiento sugerencias*** Permiten que el receptor comunique un conjunto adicional de aristas que puede ayudar al remitente a completar su pago. Las sugerencias se utilizan generalmente

cuando el receptor tiene algunos/todos los canales privados, y desea guiar al remitente a esta parte "no mapeada" del gráfico de canales. Una sugerencia de enrutamiento codifica efectivamente la misma información que un mensaje de actualización de canal normal. La actualización se empaqueta en un solo valor con los siguientes campos:

- La clave pública del nodo saliente en el borde (264 bits)
- El short_channel_id del borde "virtual" (64 bits)
- La tarifa base (fee_base_msat) del borde (32 bits)
- La tarifa proporcional (fee_proportional_millionths) (32 bits)
- El delta de caducidad de CLTV (cltv_expiry_delta) (16 bits)

La parte final del segmento de datos es el conjunto de bits de características que comunican al remitente la funcionalidad necesaria para completar un pago. Por ejemplo, si se agrega un nuevo tipo de pago en el futuro que no es compatible con el tipo de pago original, entonces el receptor puede establecer un bit de función ***requerido*** para comunicar que el pagador necesita comprender esa función para completar el pago.

Conclusión

Como hemos visto, las facturas son mucho más que una simple solicitud de cantidad. Contienen información crítica sobre ***cómo*** realizar el pago, como sugerencias de enrutamiento, la clave pública del nodo de destino, claves efímeras para aumentar la seguridad y mucho más.

Capítulo 16. Seguridad y privacidad de Lightning Network

En este capítulo, analizamos algunos de los problemas más importantes relacionados con la seguridad y la privacidad de Lightning Network. Primero, consideraremos la privacidad, lo que significa, cómo evaluarla y algunas cosas que puede hacer para proteger su propia privacidad mientras usa Lightning Network. Luego exploraremos algunos ataques comunes y técnicas de mitigación.

¿Por qué es importante la privacidad?

La propuesta de valor clave de la criptomoneda es el dinero resistente a la censura. Bitcoin ofrece a los participantes la posibilidad de almacenar y transferir su riqueza sin la interferencia de gobiernos, bancos o corporaciones. Lightning Network continúa con esta misión.

A diferencia de las soluciones de escalado triviales como los bancos de custodia de Bitcoin, Lightning Network tiene como objetivo escalar Bitcoin sin comprometer la autocustodia, lo que debería conducir a una mayor resistencia a la censura en el ecosistema de Bitcoin. Sin embargo, Lightning Network opera bajo un modelo de seguridad diferente, que presenta nuevos desafíos de seguridad y privacidad.

Definiciones de privacidad

La pregunta, "¿Es Lightning privado?" no tiene respuesta directa. La privacidad es un tema complejo; a menudo es difícil definir con precisión lo que entendemos por privacidad, especialmente si no es un investigador de privacidad. Afortunadamente, los investigadores de privacidad usan procesos para analizar y evaluar las características de privacidad de los sistemas, ¡y nosotros también podemos usarlos! Veamos cómo una seguridad

El investigador podría tratar de responder a la pregunta: "¿Es Lightning privado?" en dos pasos generales.

Primero, un investigador de privacidad definiría un ***modelo de seguridad*** que especifica lo que un adversario es capaz de hacer y pretende lograr. Luego, describirían las propiedades relevantes del sistema y verificarían si cumple con los requisitos.

Proceso para evaluar la privacidad

Un modelo de seguridad se basa en un conjunto de ***suposiciones de seguridad subyacentes.*** En los sistemas criptográficos, estas suposiciones a menudo se centran en las propiedades matemáticas de las primitivas criptográficas, como cifrados, firmas y funciones hash. Las suposiciones de seguridad de Lightning Network son que las firmas ECDSA, la función hash SHA-256 y otras funciones criptográficas utilizadas en el protocolo se comportan dentro de sus definiciones de seguridad. Por ejemplo, asumimos que es prácticamente imposible encontrar una preimagen (y una segunda preimagen) de una función hash. Esto permite que Lightning Network confíe en el mecanismo HTLC (que usa la preimagen de una función hash) para la atomicidad de los pagos multisalto: nadie excepto el destinatario final puede revelar el secreto del pago y resolver el HTLC. También asumimos un grado de conectividad en la red, es decir, que los canales Lightning forman un gráfico conectado. Por lo tanto, es posible encontrar un camino desde cualquier emisor a cualquier receptor. Finalmente, asumimos que los mensajes de red se propagan dentro de ciertos tiempos de espera.

Ahora que hemos identificado algunas de nuestras suposiciones subyacentes, consideremos algunos posibles adversarios.

Estos son algunos modelos posibles de adversarios en Lightning Network. Un nodo de reenvío "honesto pero curioso" puede observar los montos de los pagos, los nodos inmediatamente anteriores y posteriores, y el gráfico de los canales anunciados con sus capacidades. Un nodo muy bien conectado puede hacer lo mismo pero en mayor medida. Por ejemplo, considere a los desarrolladores de una billetera popular que mantienen un nodo al que sus usuarios se conectan de forma predeterminada. Este nodo sería responsable de enrutar una gran parte de los pagos hacia y desde el

usuarios de esa billetera. ¿Qué pasa si varios nodos están bajo control adversario? Si dos nodos en connivencia se encuentran en la misma ruta de pago, entenderán que están reenviando HTLC que pertenecen al mismo pago porque los HTLC tienen el mismo hash de pago.

> **NOTA**
>
> Los pagos de varias partes (consulte "Pagos de varias partes") permiten a los usuarios ofuscar los montos de sus pagos debido a sus tamaños de división no uniformes.

¿Cuáles pueden ser los objetivos de un atacante Lightning? La seguridad de la información a menudo se describe en términos de tres propiedades principales: confidencialidad, integridad y disponibilidad.

Confidencialidad

La información solo llega a los destinatarios previstos.

Integridad

La información no se altera en tránsito.

Disponibilidad

El sistema funciona la mayor parte del tiempo.

Las propiedades importantes de Lightning Network se centran principalmente en la confidencialidad y la disponibilidad. Algunas de las propiedades más importantes para proteger incluyen:

- Solo el remitente y el destinatario conocen el monto del pago.
- Nadie puede vincular emisores y receptores.
- Un usuario honesto no puede ser bloqueado para enviar y recibir pagos.

Para cada objetivo de privacidad y modelo de seguridad, existe una cierta probabilidad de que un atacante tenga éxito. Esta probabilidad depende de varios factores, como el tamaño y la estructura de la red. En igualdad de condiciones, generalmente es más fácil atacar con éxito una red pequeña que una grande. Del mismo modo, cuanto más centralizada es la red, más capaz puede ser un atacante si los nodos "centrales" están bajo su control. Por supuesto, el término centralización debe definirse con precisión para construir modelos de seguridad a su alrededor, y hay muchas definiciones posibles de qué tan centralizada es una red. Finalmente, como red de pago, Lightning Network depende de estímulos económicos. El tamaño y la estructura de las tarifas afectan el algoritmo de enrutamiento y, por lo tanto, pueden ayudar al atacante reenviando la mayoría de los pagos a través de sus nodos o evitar que esto suceda.

Conjunto de anonimato

¿Qué significa quitar el anonimato a alguien? En términos simples, la anonimización implica vincular alguna acción con la identidad del mundo real de una persona, como su nombre o dirección física. En la investigación de la privacidad, la noción de anonimización es más matizada. Primero, no estamos necesariamente hablando de nombres y direcciones. Descubrir la dirección IP o el número de teléfono de alguien también puede considerarse anonimización. Una pieza de información que permite vincular la acción de un usuario con sus acciones anteriores se conoce como ***identidad.*** En segundo lugar, la anonimización no es binaria; un usuario no es completamente anónimo ni completamente desanonimizado. En cambio, la investigación de privacidad analiza el anonimato en comparación con el conjunto de anonimato.

El ***conjunto de anonimato*** es una noción central en la investigación de la privacidad. Se refiere al conjunto de identidades tales que, desde el punto de vista de un atacante, una acción dada podría corresponder a cualquiera en el conjunto. Considere un ejemplo de la vida real. Imagina que conoces a una persona en una calle de la ciudad. ¿Cuál es su anonimato establecido desde su punto de vista? Si no los conoce personalmente y sin ninguna información adicional, su conjunto de anonimato equivale aproximadamente a la población de la ciudad, incluidos los viajeros. Si además considera su apariencia, es posible que pueda estimar aproximadamente su edad y excluir a los residentes de la ciudad que son

obviamente mayor o menor que la persona en cuestión del conjunto de anonimato. Además, si observa que la persona ingresa a la oficina de la empresa X con una credencial electrónica, el conjunto de anonimato se reduce al número de empleados y visitantes de la empresa X. Finalmente, puede notar el número de placa del automóvil que utilizaron para llegar al lugar. Si eres un observador casual, esto no te da mucho. Sin embargo, si usted es un funcionario de la ciudad y tiene acceso a la base de datos que relaciona los números de placa con los nombres, puede reducir el anonimato establecido a solo unas pocas personas: el propietario del automóvil y cualquier amigo cercano y pariente que pueda haber tomado prestado el automóvil. .

Este ejemplo ilustra algunos puntos importantes. Primero, cada bit de información puede acercar al adversario a su objetivo. Puede que no sea necesario reducir el conjunto de anonimato al tamaño de uno. Por ejemplo, si el adversario planea un ataque de denegación de servicio (DoS) dirigido y puede derribar 100 servidores, el conjunto de anonimato de 100 es suficiente. En segundo lugar, el adversario puede correlacionar información de diferentes fuentes. Incluso si una fuga de privacidad parece relativamente benigna, nunca sabemos lo que puede lograr en combinación con otras fuentes de datos. Finalmente, especialmente en configuraciones criptográficas, el atacante siempre tiene el "último recurso" de una búsqueda de fuerza bruta. Las primitivas criptográficas están diseñadas para que sea prácticamente imposible adivinar un secreto como una clave privada. Sin embargo, cada bit de información acerca al adversario a este objetivo y, en algún momento, se vuelve alcanzable.

En términos de Lightning, eliminar el anonimato generalmente significa derivar una correspondencia entre pagos y usuarios identificados por ID de nodo. A cada pago se le puede asignar un conjunto de anonimato de remitente y un conjunto de anonimato de receptor. Idealmente, el conjunto de anonimato consiste en todos los usuarios de la red. Esto asegura que el atacante no tiene información alguna. Sin embargo, la red real filtra información que permite a un atacante restringir la búsqueda. Cuanto más pequeño sea el conjunto de anonimato, mayor será la posibilidad de una desanonimización exitosa.

Diferencias entre Lightning Network y Bitcoin en términos de privacidad

Si bien es cierto que las transacciones en la red de Bitcoin no asocian identidades del mundo real con direcciones de Bitcoin, todas las transacciones se transmiten en texto no cifrado y se pueden analizar. Se han establecido varias empresas para eliminar el anonimato de los usuarios de Bitcoin y otras criptomonedas.

A primera vista, Lightning brinda una mejor privacidad que Bitcoin porque los pagos de Lightning no se transmiten a toda la red. Si bien esto mejora la línea base de privacidad, otras propiedades del protocolo Lightning pueden hacer que los pagos anónimos sean más desafiantes. Por ejemplo, los pagos más grandes pueden tener menos opciones de enrutamiento. Esto puede permitir que un adversario que controle nodos bien capitalizados enrute la mayoría de los pagos grandes y descubra los montos de los pagos y probablemente otros detalles. Con el tiempo, a medida que crece Lightning Network, esto puede convertirse en un problema menor.

Otra diferencia relevante entre Lightning y Bitcoin es que los nodos Lightning mantienen una identidad permanente, mientras que los nodos Bitcoin no. Un usuario sofisticado de Bitcoin puede cambiar fácilmente los nodos utilizados para recibir datos de blockchain y transmitir transacciones. Un usuario Lightning, por el contrario, envía y recibe pagos a través de los nodos que ha utilizado para abrir sus canales de pago. Además, el protocolo Lightning asume que los nodos de enrutamiento anuncian su dirección IP además de su ID de nodo.
Esto crea un vínculo permanente entre los ID de nodo y las direcciones IP, lo que puede ser peligroso si se tiene en cuenta que una dirección IP suele ser un paso intermedio en los ataques de anonimato vinculados a la ubicación física del usuario y, en la mayoría de los casos, a la identidad del mundo real. Es posible usar Lightning sobre Tor, pero muchos nodos no usan esta funcionalidad, como se puede ver en las estadísticas recopiladas de los anuncios de nodos.

Un usuario Lightning, al enviar un pago, tiene a sus vecinos en su conjunto de anonimato. Específicamente, un nodo de enrutamiento solo conoce los nodos inmediatamente anteriores y posteriores. El nodo de enrutamiento no sabe si sus vecinos inmediatos en la ruta de pago son el remitente final o

receptor. Por lo tanto, el conjunto de anonimato de un nodo en Lightning es aproximadamente igual al de sus vecinos (consulte la Figura 16-1).

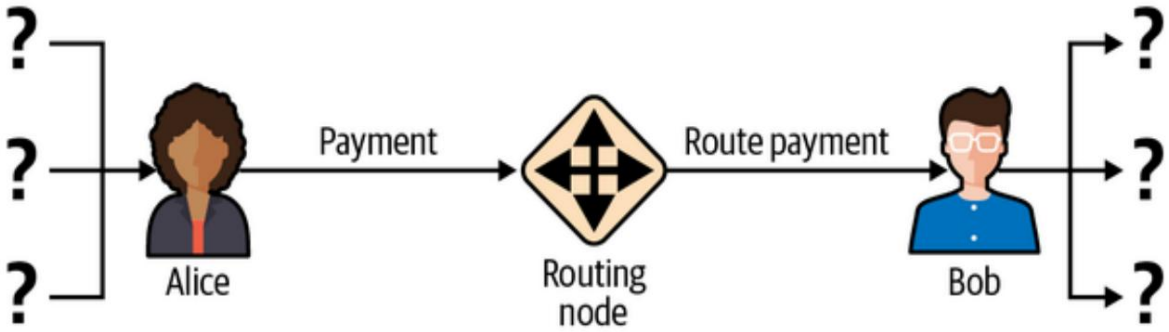

Figura 16-1. El conjunto de anonimato de Alice y Bob constituye sus vecinos

Se aplica una lógica similar a los receptores de pagos. Muchos usuarios abren solo un puñado de canales de pago, lo que limita sus conjuntos de anonimato. Además, en Lightning, el conjunto de anonimato es estático o al menos cambia lentamente. Por el contrario, uno puede lograr conjuntos de anonimato significativamente más grandes en transacciones CoinJoin en cadena. Las transacciones CoinJoin con conjuntos de anonimato mayores de 50 son bastante frecuentes. Por lo general, los conjuntos de anonimato en una transacción CoinJoin corresponden a un conjunto de usuarios que cambia dinámicamente.

Finalmente, a los usuarios de Lightning también se les puede negar el servicio, y un atacante puede bloquear o agotar sus canales. El reenvío de pagos requiere que el capital, ¡un recurso escaso!, se bloquee temporalmente en los HTLC a lo largo de la ruta.
Un atacante puede enviar muchos pagos pero no finalizarlos, ocupando el capital de los usuarios honestos durante largos períodos. Este vector de ataque no está presente (o al menos no es tan obvio) en Bitcoin.

En resumen, mientras que algunos aspectos de la arquitectura de Lightning Network sugieren que es un paso adelante en términos de privacidad en comparación con Bitcoin, otras propiedades del protocolo pueden facilitar los ataques a la privacidad.
Se necesita una investigación exhaustiva para evaluar qué garantías de privacidad proporciona Lightning Network y mejorar la situación.

Los temas discutidos en esta parte del capítulo resumen la investigación disponible a mediados de 2021. Sin embargo, esta área de investigación y desarrollo está creciendo rápidamente. Nos complace informar que los autores conocen varios equipos de investigación que trabajan actualmente en la privacidad de Lightning.

Ahora revisemos algunos de los ataques a la privacidad de LN que se han descrito en la literatura académica.

Ataques a relámpagos

Investigaciones recientes describen varias formas en las que la seguridad y la privacidad de Lightning Network pueden verse comprometidas.

Observación de montos de pago

Uno de los objetivos de un sistema de pago que preserva la privacidad es ocultar el monto del pago a las partes no involucradas. Lightning Network es una mejora sobre la Capa 1 en este sentido. Si bien las transacciones de Bitcoin se transmiten en texto sin cifrar y cualquier persona puede observarlas, los pagos Lightning solo viajan a través de unos pocos nodos a lo largo de la ruta de pago. Sin embargo, los nodos intermediarios sí ven el monto del pago, aunque este monto del pago puede no corresponder al monto del pago total real (consulte “Pagos de varias partes”). Esto es necesario para crear un nuevo HTLC en cada salto. La disponibilidad de montos de pago para los nodos intermediarios no presenta una amenaza inmediata. Sin embargo, un nodo intermediario ***honesto pero curioso*** puede usarlo como parte de un ataque más grande.

Vinculación de remitentes y receptores

Un atacante podría estar interesado en conocer el remitente y/o el receptor de un pago para revelar ciertas relaciones económicas. Esta violación de la privacidad podría dañar la resistencia a la censura, ya que un nodo intermediario podría censurar los pagos hacia o desde ciertos destinatarios o remitentes. Idealmente, la vinculación de remitentes con receptores no debería ser posible para nadie más que el remitente y el receptor.

En las siguientes secciones, consideraremos dos tipos de adversarios: el adversario fuera del camino y el adversario en el camino. Un adversario fuera de la ruta intenta evaluar al remitente y al receptor de un pago sin participar en el

proceso de enrutamiento de pago. Un adversario en camino puede aprovechar cualquier información que pueda obtener enrutando el pago de intereses.

Primero, considere al ***adversario fuera del camino.*** En el primer paso de este escenario de ataque, un potente adversario fuera de ruta deduce los saldos individuales en cada canal de pago a través de un sondeo (descrito en una sección posterior) y forma una instantánea de la red en el momento t_1, que será las 12:05. Luego, el atacante hará sondear la red en algún momento posterior en el tiempo t_2, que será 12:10. Luego, el atacante compararía las instantáneas a las 12:10 y las 12:05 y usaría las diferencias entre las dos instantáneas para inferir información sobre los pagos que se realizaron al observar las rutas que cambiaron. En el caso más simple, si solo se produjera un pago entre las 12:10 y las 12:05, el adversario observaría un único camino donde los saldos han cambiado en las mismas cantidades. Así, el adversario aprende casi todo sobre este pago: el remitente, el destinatario y el monto. Si varias rutas de pago se superponen, el adversario debe aplicar heurísticas para identificar dicha superposición y separar los pagos.

Ahora, dirigimos nuestra atención a un ***adversario en camino.*** Tal adversario puede parecer complicado. Sin embargo, en junio de 2020, los investigadores notaron que el nodo más central observó cerca del 50 % de todos los pagos de LN, mientras que los cuatro nodos más centrales observaron un promedio de 72% de pagos. Estos hallazgos enfatizan la relevancia del modelo de atacante en ruta. Aunque los intermediarios en una ruta de pago solo conocen a su sucesor y predecesor, existen varias filtraciones que un intermediario malicioso u honesto, pero curioso, podría usar para inferir el remitente y el receptor.

El adversario en ruta puede observar el monto de cualquier pago enrutado, así como los deltas de bloqueo de tiempo (consulte el Capítulo 10). Por lo tanto, el adversario puede excluir cualquier nodo del conjunto de anonimato del remitente o del receptor con capacidades inferiores a la cantidad enrutada. Por lo tanto, observamos una compensación entre privacidad y montos de pago. Por lo general, cuanto mayor es el monto del pago, más pequeños son los conjuntos de anonimato. Observamos que esta fuga podría minimizarse con pagos multiparte o con canales de pago de gran capacidad. De manera similar, los canales de pago con pequeños deltas de bloqueo de tiempo podrían excluirse de una ruta de pago. Más precisamente, un canal de pago no puede

pertenecen a un pago si el tiempo restante durante el cual el pago podría estar bloqueado es mayor que lo que el nodo de reenvío estaría dispuesto a aceptar. Esta fuga podría ser desalojada adhiriéndose a las llamadas rutas sombra.

Una de las filtraciones más sutiles y poderosas que un adversario en camino puede fomentar es el análisis de tiempo. Un adversario en ruta puede mantener un registro de cada pago enrutado, junto con la cantidad de tiempo que tarda un nodo en responder a una solicitud HTLC. Antes de comenzar el ataque, el atacante aprende las características de latencia de cada nodo en Lightning Network enviándoles solicitudes. Naturalmente, esto puede ayudar a establecer la posición precisa del adversario en la ruta de pago. Más aún, como se demostró recientemente, un atacante puede determinar con éxito el remitente y el destinatario de un pago a partir de un conjunto de posibles remitentes y destinatarios utilizando estimadores basados en el tiempo.

Finalmente, es importante reconocer que probablemente existan filtraciones desconocidas o no estudiadas que podrían ayudar a los intentos de anonimización. Por ejemplo, debido a que diferentes billeteras Lightning aplican diferentes algoritmos de enrutamiento, incluso conocer el algoritmo de enrutamiento aplicado podría ayudar a excluir ciertos nodos de ser un remitente y/o receptor de un pago.

Revelación de saldos de canales (sondeo)

Se supone que los saldos de los canales Lightning están ocultos por razones de privacidad y eficiencia. Un nodo Lightning solo conoce los saldos de sus canales adyacentes. El protocolo no proporciona una forma estándar de consultar el saldo de un canal remoto.

Sin embargo, un atacante puede revelar el saldo de un canal remoto en un ***ataque de sondeo.*** En seguridad de la información, el sondeo se refiere a la técnica de enviar solicitudes a un sistema objetivo y sacar conclusiones sobre su estado privado en función de las respuestas recibidas.

Los canales de rayos son propensos a sondear. Recuerde que un pago Lightning estándar comienza cuando el receptor crea un secreto de pago aleatorio y envía su hash al remitente. Tenga en cuenta que para los nodos intermediarios, todos

los hash parecen aleatorios. No hay forma de saber si un hash corresponde a un secreto real o si se generó aleatoriamente.

El ataque de sondeo procede de la siguiente manera. Digamos que el atacante Mallory quiere revelar el saldo de Alice de un canal público entre Alice y Bob. Supongamos que la capacidad total de ese canal es de 1 millón de satoshis. El saldo de Alice puede oscilar entre cero y 1 millón de satoshis (para ser precisos, la estimación es un poco más ajustada debido a la reserva de canales, pero no la tomamos en cuenta aquí por simplicidad). Mallory abre un canal con Alice con 1 millón de satoshis y envía 500 000 satoshis a Bob a través de Alice utilizando un ***número aleatorio*** como hash de pago. Por supuesto, este número no corresponde a ningún secreto de pago conocido. Por lo tanto, el pago fallará. La pregunta es: ¿cómo fallará exactamente?

Hay dos escenarios. Si Alice tiene más de 500 000 satoshis en su lado del canal para Bob, ella reenvía el pago. Bob descifra la cebolla de pago y se da cuenta de que el pago está destinado a él. Busca en su tienda local de secretos de pago y busca la preimagen que corresponde al hash de pago, pero no la encuentra. Siguiendo el protocolo, Bob devuelve el error de "hash de pago desconocido" a Alice, quien se lo transmite a Mallory. Como resultado, Mallory sabe que el pago ***podría haberse realizado correctamente*** si el hash del pago fuera real. Por lo tanto, Mallory puede actualizar su estimación del saldo de Alice de “entre cero y 1 millón” a “entre 500 000 y 1 millón”. Otro escenario ocurre si el saldo de Alice es inferior a 500.000 satoshis. En ese caso, Alice no puede reenviar el pago y devuelve el error de "saldo insuficiente" a Mallory.

Mallory actualiza su estimación de “entre cero y 1 millón” a “entre cero y 500 000”.

Tenga en cuenta que, en cualquier caso, la estimación de Mallory se vuelve el doble de precisa después de un solo sondeo. Puede continuar sondeando, eligiendo la siguiente cantidad de sondeo de modo que divida el intervalo de estimación actual por la mitad. Esta conocida técnica de búsqueda se llama ***búsqueda binaria.*** Con la búsqueda binaria, el número de sondas es ***logarítmico*** en la precisión deseada. Por ejemplo, para obtener el saldo de Alice en un canal de 1 millón de satoshis hasta un solo satoshi, Mallory solo tendría que realizar log2 (1,000,000) ÿ 20

sondeos Si un sondeo tarda 3 segundos, ¡un canal se puede sondear con precisión en solo un minuto!

El sondeo de canales se puede hacer aún más eficiente. En su variante más simple, Mallory se conecta directamente al canal que quiere sondear. ¿Es posible sondear un canal sin abrir un canal a uno de sus puntos finales? Imagine que Mallory ahora quiere probar un canal entre Bob y Charlie, pero no quiere abrir otro canal, lo que requiere pagar tarifas en cadena y esperar confirmaciones de las transacciones de financiación. En su lugar, Mallory reutiliza su canal existente a Alice y envía una sonda a lo largo de la ruta Mallory ÿ Alice ÿ Bob ÿ Charlie. Mallory puede interpretar el error de "hash de pago desconocido" de la misma manera que antes: la sonda ha llegado al destino; por lo tanto, todos los canales a lo largo de la ruta tienen saldos suficientes para reenviarlo. Pero, ¿y si Mallory recibe el error de "saldo insuficiente"?

¿Significa que el equilibrio es insuficiente entre Alice y Bob o entre Bob y Charlie?

En el protocolo Lightning actual, los mensajes de error informan no solo ***qué*** error ocurrió sino también ***dónde*** ocurrió. Entonces, con un manejo de errores más cuidadoso, Mallory ahora sabe qué canal falló. Si este es el canal objetivo, actualiza sus estimaciones; si no, elige otra ruta hacia el canal de destino. Incluso obtiene información ***adicional*** sobre los saldos de los canales intermediarios, además del canal de destino.

El ataque de sondeo se puede utilizar además para vincular remitentes y receptores, como se describe en la sección anterior.

En este punto, puede preguntarse: ¿por qué Lightning Network hace un trabajo tan pobre en la protección de los datos privados de sus usuarios? ¿No sería mejor no revelar al remitente por qué y dónde ha fallado el pago? De hecho, esto podría ser una contramedida potencial, pero tiene importantes inconvenientes. Lightning tiene que lograr un cuidadoso equilibrio entre privacidad y eficiencia. Recuerde que los nodos regulares no conocen las distribuciones de saldos en los canales remotos.
Por lo tanto, los pagos pueden fallar (y a menudo lo hacen) debido a un saldo insuficiente en un salto intermediario. Los mensajes de error permiten al remitente excluir el canal que falla al construir otra ruta. Una

La popular billetera Lightning incluso realiza un sondeo interno para verificar si una ruta construida realmente puede manejar un pago.

Existen otras contramedidas potenciales contra el sondeo de canales. Primero, es difícil para un atacante apuntar a canales no anunciados. En segundo lugar, los nodos que implementan enrutamiento justo a tiempo (JIT) pueden ser menos propensos al ataque.
Finalmente, dado que los pagos de varias partes hacen que el problema de la capacidad insuficiente sea menos grave, los desarrolladores del protocolo pueden considerar ocultar algunos de los detalles del error sin dañar la eficiencia.

Negación de servicio

Cuando los recursos se ponen a disposición del público, existe el riesgo de que los atacantes intenten hacer que ese recurso no esté disponible mediante la ejecución de un ataque de denegación de servicio (DoS). Generalmente, esto se logra cuando el atacante bombardea un recurso con solicitudes, que son indistinguibles de las consultas legítimas. Los ataques rara vez dan como resultado que el objetivo sufra pérdidas financieras, aparte del costo de oportunidad de la caída de su servicio, y simplemente tienen la intención de agraviar al objetivo.

Las mitigaciones típicas de los ataques DoS requieren la autenticación de las solicitudes para separar a los usuarios legítimos de los malintencionados. Estas mitigaciones incurren en un costo trivial para los usuarios regulares, pero actuarán como un impedimento suficiente para que un atacante inicie solicitudes a gran escala. Las medidas contra la denegación de servicio se pueden ver en todas partes en Internet: los sitios web aplican límites de velocidad para garantizar que ningún usuario pueda consumir toda la atención de su servidor, los sitios de revisión de películas requieren autenticación de inicio de sesión para mantener enojado r/prequelmemes (grupo Reddit) miembros a raya, y los servicios de datos venden claves API para limitar el número de consultas.

DoS en bitcoin

En Bitcoin, el ancho de banda que utilizan los nodos para transmitir transacciones y el espacio que aprovechan para la red en forma de su mempool son recursos disponibles públicamente. Cualquier nodo de la red puede consumir ancho de banda y espacio de mempool enviando una transacción válida. Si esta transacción es minada

en un bloque válido, pagarán tarifas de transacción, lo que agrega un costo al uso de estos recursos de red compartidos.

En el pasado, la red Bitcoin se enfrentó a un intento de ataque DoS en el que los atacantes enviaron spam a la red con transacciones de bajo costo. Muchas de estas transacciones no fueron seleccionadas por los mineros debido a sus bajas tarifas de transacción, por lo que los atacantes podían consumir recursos de la red sin pagar las tarifas. Para abordar este problema, se estableció una tarifa mínima de retransmisión de transacciones que establece una tarifa de umbral que los nodos requieren para propagar transacciones. Esta medida aseguró en gran medida que las transacciones que consumen recursos de la red finalmente pagarán sus tarifas de cadena. La tarifa mínima de retransmisión es aceptable para los usuarios habituales, pero perjudicaría financieramente a los atacantes si intentaran enviar spam a la red. Si bien es posible que algunas transacciones no se conviertan en bloques válidos en entornos de tarifas altas, estas medidas han sido en gran medida efectivas para disuadir este tipo de spam.

DoS en relámpagos

De manera similar a Bitcoin, Lightning Network cobra tarifas por el uso de sus recursos públicos, pero en este caso, los recursos son canales públicos y las tarifas vienen en forma de tarifas de enrutamiento. La capacidad de enrutar pagos a través de nodos a cambio de tarifas brinda a la red un gran beneficio de escalabilidad (los nodos que no están conectados directamente aún pueden realizar transacciones), pero tiene el costo de exponer un recurso público que debe protegerse contra ataques DoS.

Cuando un nodo Lightning reenvía un pago en su nombre, utiliza datos y ancho de banda de pago para actualizar su transacción de compromiso, y el monto del pago se reserva en el saldo de su canal hasta que se liquide o falle. En pagos exitosos, esto es aceptable porque el nodo finalmente paga sus tarifas. Los pagos fallidos no incurren en cargos en el protocolo actual. Esto permite que los nodos enruten sin costo los pagos fallidos a través de cualquier canal. Esto es excelente para los usuarios legítimos, a quienes no les gustaría pagar por los intentos fallidos, pero también les permite a los atacantes consumir los recursos de los nodos sin costo, al igual que las transacciones de bajo costo en Bitcoin que nunca terminan pagando las tarifas de los mineros.

En el momento de escribir este artículo, hay una discusión en curso . en la lista de correo de lightning-dev sobre la mejor manera de abordar este problema.

Ataques DoS conocidos

Hay dos ataques DoS conocidos en canales LN públicos que inutilizan un canal de destino, o un conjunto de canales de destino. Ambos ataques implican el enrutamiento de pagos a través de un canal público y luego retenerlos hasta su tiempo de espera, lo que maximiza la duración del ataque. El requisito de fallar en los pagos para no pagar las tarifas es bastante simple de cumplir porque los nodos maliciosos pueden simplemente redirigir los pagos hacia ellos mismos. En ausencia de tarifas por pagos fallidos, el único costo para el atacante es el costo en cadena de abrir un canal para enviar estos pagos, lo que puede ser trivial en entornos de tarifas bajas.

Interferencia de compromiso

Los nodos Lightning actualizan su estado compartido mediante transacciones de compromiso asimétricas, en las que se agregan y eliminan HTLC para facilitar los pagos. Cada partido está limitado a un total de 483 HTLC en la transacción de compromiso a la vez. Un ataque de interferencia de canal permite que un atacante inutilice un canal enrutando pagos 483 a través del canal de destino y reteniéndolos hasta que se agote el tiempo de espera.

Cabe señalar que este límite se eligió en la especificación para garantizar que todos los HTLC se puedan barrer en una sola transacción judicial. Si bien este límite ***puede*** aumentar, las transacciones aún están limitadas por el tamaño del bloque, por lo que es probable que la cantidad de espacios disponibles siga siendo limitada.

Bloqueo de liquidez del canal

Un ataque de bloqueo de liquidez del canal es comparable a un ataque de bloqueo del canal en el sentido de que enruta los pagos a través de un canal y los retiene para que el canal quede inutilizable. En lugar de bloquear espacios en el compromiso del canal, este ataque enruta grandes HTLC a través de un canal de destino, consumiendo todo el ancho de banda disponible del canal. La capital de este ataque

el compromiso es más alto que el ataque de interferencia de compromiso porque el nodo atacante necesita más fondos para enrutar los pagos fallidos a través del objetivo.

Desanonimización de capas cruzadas

Las redes informáticas suelen estar en capas. La estratificación permite la separación de preocupaciones y hace que todo el sistema sea manejable. Nadie podría diseñar un sitio web si requiriera comprender toda la pila de TCP/IP hasta la codificación física de bits en un cable óptico. Se supone que cada capa proporciona la funcionalidad a la capa superior de una manera limpia. Idealmente, la capa superior debería percibir una capa inferior como una caja negra. Sin embargo, en realidad, las implementaciones no son ideales y los detalles se ***filtran*** a la capa superior. Este es el problema de las abstracciones con fugas.

En el contexto de Lightning, el protocolo LN se basa en el protocolo Bitcoin y la red LN P2P. Hasta este punto, solo consideramos las garantías de privacidad que ofrece Lightning Network de forma aislada. Sin embargo, la creación y el cierre de canales de pago se realizan inherentemente en la cadena de bloques de Bitcoin. En consecuencia, para un análisis completo de las disposiciones de privacidad de Lightning Network, es necesario considerar cada capa de la pila tecnológica con la que los usuarios podrían interactuar. Específicamente, un adversario anónimo puede usar y usará datos fuera y dentro de la cadena para agrupar o vincular nodos LN a las direcciones de Bitcoin correspondientes.

Los atacantes que intentan eliminar el anonimato de los usuarios de LN pueden tener varios objetivos, en un contexto de capas cruzadas:

- Clúster de direcciones Bitcoin propiedad del mismo usuario (Capa 1). Llamamos a estas entidades Bitcoin.
- Nodos de clúster LN propiedad del mismo usuario (capa 2).
- Vincule sin ambigüedades conjuntos de nodos LN a los conjuntos de entidades Bitcoin que los controlan.

Hay varias heurísticas y patrones de uso que permiten a un adversario agrupar direcciones de Bitcoin y nodos de LN propiedad de los mismos usuarios de LN.
Además, estos clústeres se pueden vincular a través de capas utilizando otras potentes heurísticas de vinculación entre capas. El último tipo de heurística, las técnicas de enlace entre capas, enfatiza la necesidad de una visión holística de la privacidad. Específicamente, debemos considerar la privacidad en el contexto de ambas capas juntas.

Agrupación de entidades de Bitcoin en cadena

Las interacciones de la cadena de bloques Lightning Network se reflejan permanentemente en el gráfico de entidades de Bitcoin. Incluso si un canal está cerrado, un atacante puede observar qué dirección financió el canal y dónde se gastaron las monedas después de cerrarlo. Para este análisis, consideremos cuatro entidades separadas. La apertura de un canal provoca un flujo monetario de una entidad de ***origen*** a una ***entidad de financiación;*** cerrar un canal provoca un flujo desde una entidad de ***liquidación*** a una ***entidad de destino.***

A principios de 2021, Romiti et al. identificó cuatro heurísticas que permiten la agrupación de estas entidades. Dos de ellos capturan cierto comportamiento de financiación con fugas y dos describen comportamientos de liquidación con fugas.

Estrella heurística (financiación)

> Si un componente contiene una entidad de origen que envía fondos a una o más entidades de financiación, es probable que estas entidades de financiación estén controladas por el mismo usuario.

Serpiente heurística (financiamiento)

> Si un componente contiene una entidad de origen que reenvía fondos a una o más entidades, que a su vez se utilizan como entidades de origen y financiación, es probable que todas estas entidades estén controladas por el mismo usuario.

Recopilador heurístico (liquidación)

> Si un componente contiene una entidad de destino que recibe fondos de una o más entidades de liquidación, estas entidades de liquidación probablemente estén controladas por el mismo usuario.

Proxy heurístico (liquidación)

> Si un componente contiene una entidad de destino que recibe fondos de una o más entidades, que a su vez se utilizan como entidades de liquidación y destino, es probable que estas entidades estén controladas por el mismo usuario.

Vale la pena señalar que estas heurísticas pueden producir falsos positivos. Por ejemplo, si las transacciones de varios usuarios no relacionados se combinan en una transacción CoinJoin, entonces la estrella o la heurística de proxy pueden producir falsos positivos. Esto podría suceder si los usuarios están financiando un canal de pago a partir de una transacción CoinJoin. Otra fuente potencial de falsos positivos podría ser que una entidad pudiera representar a varios usuarios si las direcciones agrupadas están controladas por un servicio (por ejemplo, intercambio) o en nombre de sus usuarios (cartera de custodia). Sin embargo, estos falsos positivos se pueden filtrar de manera efectiva.

contramedidas

Si los resultados de las transacciones de financiación no se reutilizan para abrir otros canales, la heurística de la serpiente no funciona. Si los usuarios se abstienen de utilizar canales de financiación de una única fuente externa y evitan recaudar fondos en una única entidad de destino externa, las otras heurísticas no arrojarían ningún resultado significativo.

Agrupación de nodos Lightning fuera de la cadena

Los nodos de LN anuncian alias, por ejemplo, ***LNBig.com.*** Los alias pueden mejorar la usabilidad del sistema. Sin embargo, los usuarios tienden a usar alias similares para sus propios nodos diferentes. Por ejemplo, es probable que ***LNBig.com Billing*** sea propiedad del mismo usuario que el nodo con el alias ***LNBig.com.*** Dada esta observación, uno puede agrupar nodos LN aplicando sus alias de nodo. Específicamente, uno agrupa los nodos LN en una sola dirección si sus alias son similares con respecto a alguna métrica de similitud de cadenas.

Otro método para agrupar nodos LN es aplicar sus direcciones IP o Tor. Si las mismas direcciones IP o Tor corresponden a diferentes nodos LN, es probable que estos nodos estén controlados por el mismo usuario.

contramedidas

Para mayor privacidad, los alias deben ser lo suficientemente diferentes entre sí.
Si bien el anuncio público de direcciones IP puede ser inevitable para aquellos nodos que desean tener canales entrantes en Lightning Network, la capacidad de vinculación entre nodos del mismo usuario puede mitigarse si los clientes de cada nodo están alojados con diferentes proveedores de servicios y, por lo tanto, direcciones IP. .

Enlace entre capas: Lightning Nodes y Bitcoin Entidades

Asociar nodos LN a entidades Bitcoin es una violación grave de la privacidad que se ve agravada por el hecho de que la mayoría de los nodos LN exponen públicamente sus direcciones IP. Por lo general, una dirección IP se puede considerar como un identificador único de un usuario. Dos patrones de comportamiento ampliamente observados revelan vínculos entre los nodos LN y las entidades de Bitcoin:

reutilización de monedas

> Cada vez que los usuarios cierran los canales de pago, recuperan sus monedas correspondientes. Sin embargo, muchos usuarios reutilizan esas monedas para abrir un nuevo canal. Esas monedas se pueden vincular efectivamente a un nodo LN común.

Reutilización de entidades

> Por lo general, los usuarios financian sus canales de pago desde direcciones de Bitcoin correspondientes a la misma entidad de Bitcoin.

Estos algoritmos de vinculación de capas cruzadas podrían frustrarse si los usuarios poseen múltiples direcciones no agrupadas o usan múltiples billeteras para interactuar con Lightning Network.

La posible anonimización de las entidades de Bitcoin ilustra lo importante que es considerar la privacidad de ambas capas simultáneamente en lugar de una a la vez.

Gráfico de relámpagos

Lightning Network, como sugiere su nombre, es una red de canales de pago entre pares. Por lo tanto, muchas de sus propiedades (privacidad, robustez, conectividad, eficiencia de enrutamiento) están influenciadas y caracterizadas por su naturaleza de red.

En esta sección, discutimos y analizamos Lightning Network desde el punto de vista de la ciencia de redes. Estamos particularmente interesados en comprender el gráfico de canal LN, su robustez, conectividad y otras características importantes.

¿Cómo se ve el gráfico de rayos en la realidad?

Uno podría haber esperado que Lightning Network sea un gráfico aleatorio, donde los bordes se forman aleatoriamente entre los nodos. Si este fuera el caso, entonces la distribución de grados de Lightning Network seguiría una distribución normal gaussiana. En particular, la mayoría de los nodos tendrían aproximadamente el mismo grado y no esperaríamos nodos con grados extraordinariamente grandes. Esto se debe a que la distribución normal disminuye exponencialmente para valores fuera del intervalo alrededor del valor promedio de la distribución. La representación de un gráfico aleatorio (como vimos en la Figura 12-2) parece una topología de red en malla. Parece descentralizado y no jerárquico: cada nodo parece tener la misma importancia. Además, los gráficos aleatorios tienen un gran diámetro. En particular, el enrutamiento en dichos gráficos es un desafío porque el camino más corto entre dos nodos cualesquiera es moderadamente largo.

Sin embargo, en marcado contraste, el gráfico LN es completamente diferente.

Gráfico de relámpagos hoy

Lightning es una red financiera. Así, el crecimiento y la formación de la red también están influenciados por incentivos económicos. Cada vez que un nodo se une a Lightning Network, es posible que desee maximizar su conectividad con otros nodos para aumentar su eficiencia de enrutamiento. Este fenómeno se llama apego preferencial. Estos incentivos económicos dan como resultado una red fundamentalmente diferente a un gráfico aleatorio.

Basado en instantáneas de canales anunciados públicamente, la distribución de grados de Lightning Network sigue una función de ley de potencia. En dicho gráfico, la gran mayoría de los nodos tienen muy pocas conexiones con otros nodos, mientras que solo unos pocos nodos tienen numerosas conexiones. En un nivel alto, esta topología gráfica se parece a una estrella: la red tiene un núcleo bien conectado y una periferia débilmente conectada. Las redes con distribución de grado de ley de potencia también se denominan redes sin escala. Esta topología es ventajosa para enrutar pagos de manera eficiente, pero es propensa a ciertos ataques basados en topología.

Ataques basados en topología

Un adversario podría querer interrumpir Lightning Network y decidir que su objetivo es desmantelar toda la red en muchos componentes más pequeños, haciendo que el enrutamiento de pagos sea prácticamente imposible en toda la red. Un objetivo menos ambicioso, pero aún malicioso y severo, podría ser solo eliminar ciertos nodos de la red. Tal interrupción puede ocurrir en el nivel de nodo o en el nivel de borde.

Supongamos que un adversario puede derribar cualquier nodo en Lightning Network. Por ejemplo, puede atacarlos con un ataque de denegación de servicio distribuido (DDoS) o hacerlos no operativos por cualquier medio. Resulta que si el adversario elige nodos al azar, las redes libres de escala como Lightning Network son sólidas contra los ataques de eliminación de nodos. Esto se debe a que un nodo aleatorio se encuentra en la periferia con una pequeña cantidad de conexiones, por lo que juega un papel insignificante en la conectividad de la red. Sin embargo, si el adversario es más prudente, puede apuntar a los nodos mejor conectados. No es sorprendente que Lightning Network y

otras redes libres de escala ***no*** son resistentes frente a ataques dirigidos de eliminación de nodos.

Por otro lado, el adversario podría ser más sigiloso. Varios ataques basados en topología tienen como objetivo un solo nodo o un solo canal de pago. Por ejemplo, un adversario podría estar interesado en agotar la capacidad de un determinado canal de pago a propósito. En términos más generales, un adversario puede agotar toda la capacidad de salida de un nodo para eliminarlo del mercado de enrutamiento.
Esto podría obtenerse fácilmente enrutando los pagos a través del nodo víctima con montos equivalentes a la capacidad de salida de cada canal de pago.
Después de completar este llamado ataque de aislamiento de nodos, la víctima ya no puede enviar ni enrutar pagos a menos que reciba un pago o reequilibre sus canales.

Para concluir, incluso por diseño, es posible eliminar bordes y nodos de la Lightning Network enrutable. Sin embargo, dependiendo del vector de ataque utilizado, el adversario puede tener que proporcionar más o menos recursos para llevar a cabo el ataque.

Temporalidad de la Lightning Network

Lightning Network es una red sin permiso que cambia dinámicamente.
Los nodos pueden unirse o abandonar libremente la red, pueden abrir y crear canales de pago en cualquier momento que lo deseen. Por lo tanto, una sola instantánea estática del gráfico LN es engañosa. Necesitamos considerar la temporalidad y la naturaleza siempre cambiante de la red. Por ahora, el gráfico LN está creciendo en términos de número de nodos y canales de pago. Su diámetro efectivo también se está reduciendo; es decir, los nodos se vuelven más cercanos entre sí, como podemos ver en la Figura 16-2.

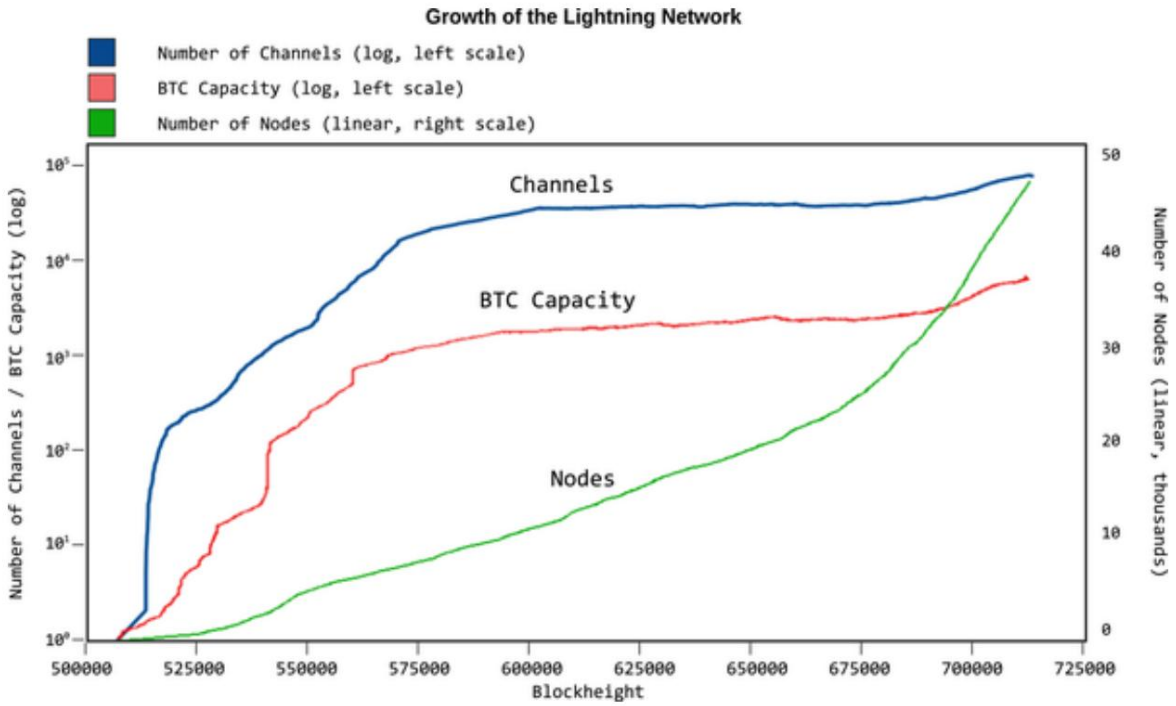

Figura 16-2. El crecimiento constante de Lightning Network en nodos, canales y capacidad bloqueada (a partir de septiembre de 2021)

En las redes sociales, el comportamiento de cierre de triángulos es común. Específicamente, en un gráfico donde los nodos representan personas y las amistades se representan como bordes, se espera que surjan triángulos en el gráfico. Un triángulo, en este caso, representa amistades por parejas entre tres personas. Por ejemplo, si Alice conoce a Bob y Bob conoce a Charlie, es probable que en algún momento Bob le presente a Alice a Charlie. Sin embargo, este comportamiento sería extraño en Lightning Network. Los nodos simplemente no están incentivados para cerrar triángulos porque podrían haber enrutado los pagos en lugar de abrir un nuevo canal de pago. Sorprendentemente, el cierre de triángulos es una práctica común en Lightning Network. El número de triángulos crecía constantemente antes de la implementación de los pagos en varias partes. Esto es contrario a la intuición y sorprendente dado que los nodos podrían haber enrutado los pagos a través de los dos lados del triángulo en lugar de abrir el tercer canal. Esto puede significar que las ineficiencias en el enrutamiento incentivaron a los usuarios a cerrar triángulos y no recurrir al enrutamiento. Con suerte, los pagos de varias partes ayudarán a aumentar la eficacia del enrutamiento de pagos.

Centralización en Lightning Network

Una métrica común para evaluar la centralidad de un nodo en un gráfico es su ***centralidad de intermediación.*** El dominio del punto central es una métrica derivada de la centralidad de intermediación, que se utiliza para evaluar la centralidad de una red. Para una definición precisa de la dominancia del punto central, se remite al lector al trabajo de Freeman.

Cuanto mayor sea el dominio del punto central de una red, más centralizada será la red. Podemos observar que Lightning Network tiene un mayor dominio del punto central (es decir, está más centralizado) que un gráfico aleatorio (gráfico Erdÿs-Rényi) o un gráfico sin escala (gráfico Barabási-Albert) de igual tamaño.

En general, nuestra comprensión de la naturaleza dinámica del gráfico de canales LN es bastante limitada. Es fructífero analizar cómo los cambios de protocolo, como los pagos de varias partes, pueden afectar la dinámica de Lightning Network. Sería beneficioso explorar la naturaleza temporal del gráfico LN con más profundidad.

Incentivos económicos y estructura gráfica

El gráfico LN se forma espontáneamente y los nodos se conectan entre sí en función del interés mutuo. Como resultado, los incentivos impulsan el desarrollo de gráficos. Veamos algunos de los incentivos relevantes:

- Incentivos racionales:
  - Los nodos establecen canales para enviar, recibir y enrutar pagos (ganar tarifas).
  - ¿Qué hace más probable que se establezca un canal entre dos nodos que actúan racionalmente?
- Incentivos altruistas:
  - Los nodos establecen canales “por el bien de la red”.

- Si bien no debemos basar nuestras suposiciones de seguridad en el altruismo, hasta cierto punto, el comportamiento altruista impulsa a Bitcoin (aceptar conexiones entrantes, servir bloques).
- ¿Qué papel juega en Lightning?

En las primeras etapas de Lightning Network, muchos operadores de nodos afirmaron que las tarifas de enrutamiento ganadas no compensan los costos de oportunidad derivados del bloqueo de liquidez. Esto indicaría que operar un nodo puede estar impulsado principalmente por incentivos altruistas "por el bien de la red". Esto podría cambiar en el futuro si Lightning Network tiene un tráfico significativamente mayor o si surge un mercado de tarifas de enrutamiento. Por otro lado, si un nodo desea optimizar sus tarifas de enrutamiento, minimizaría las longitudes de ruta más cortas promedio a todos los demás nodos. Dicho de otra manera, un nodo en busca de ganancias intentará ubicarse en el ***centro*** del gráfico del canal o cerca de él.

Consejos prácticos para que los usuarios protejan sus Privacidad

Todavía estamos en las primeras etapas de Lightning Network. Es probable que muchas de las preocupaciones enumeradas en este capítulo se aborden a medida que madure y crezca. Mientras tanto, hay algunas medidas que puede tomar para proteger su nodo contra usuarios maliciosos; algo tan simple como actualizar los parámetros predeterminados con los que se ejecuta su nodo puede ser de gran ayuda para fortalecer su nodo.

Canales no anunciados

Si tiene la intención de usar Lightning Network para enviar y recibir fondos entre nodos y billeteras que controla, y no tiene interés en enrutar los pagos de otros usuarios, no hay necesidad de anunciar sus canales al resto de la red. Podría abrir un canal entre, por ejemplo, su PC de escritorio que ejecuta un nodo completo y su teléfono móvil que ejecuta una billetera Lightning, y

simplemente omita el anuncio del canal discutido en el Capítulo 3. Estos a veces se denominan canales "privados"; sin embargo, es más correcto referirse a ellos como canales “no anunciados” porque no son estrictamente privados.

Los canales no anunciados no serán conocidos por el resto de la red y normalmente no se utilizarán para enrutar los pagos de otros usuarios. Todavía se pueden usar para enrutar pagos si otros nodos se dan cuenta de ellos; por ejemplo, una factura podría contener sugerencias de enrutamiento que sugieran una ruta con un canal no anunciado. Sin embargo, suponiendo que solo haya abierto un canal sin previo aviso con usted mismo, obtendrá cierta medida de privacidad. Dado que no está exponiendo su canal a la red, reduce el riesgo de un ataque de denegación de servicio en su nodo. También puedes administrar más fácilmente la capacidad de este canal, ya que solo se utilizará para recibir o enviar directamente a tu nodo.

También hay ventajas en abrir un canal no anunciado con una parte conocida con la que realiza transacciones con frecuencia. Por ejemplo, si Alice y Bob juegan con frecuencia al póquer por bitcoins, podrían abrir un canal para enviar sus ganancias de un lado a otro. En condiciones normales, este canal no se utilizará para enrutar pagos de otros usuarios o cobrar tarifas. Y dado que el canal no será conocido por el resto de la red, los pagos entre Alice y Bob no se pueden inferir mediante el seguimiento de los cambios en la capacidad de enrutamiento del canal. Esto confiere cierta privacidad a Alice y Bob; sin embargo, si uno de ellos decide hacer que otros usuarios conozcan el canal, por ejemplo, incluyéndolo en las sugerencias de enrutamiento de una factura, entonces se pierde esta privacidad.

También se debe tener en cuenta que para abrir un canal no anunciado, se debe realizar una transacción pública en la cadena de bloques de Bitcoin. Por lo tanto, es posible inferir la existencia y el tamaño del canal si una parte malintencionada está monitoreando la cadena de bloques en busca de transacciones de apertura de canales e intentando relacionarlas con los canales de la red. Además, cuando se cierre el canal, el saldo final del canal se hará público una vez que esté comprometido con la cadena de bloques de Bitcoin. Sin embargo, dado que las transacciones de apertura y compromiso son seudónimas, no será sencillo volver a conectarlo con Alice o Bob. Además, la actualización de Taproot de 2021 lo hace

difícil distinguir entre transacciones de apertura y cierre de canales y otros tipos específicos de transacciones de Bitcoin. Por lo tanto, si bien los canales no anunciados no son completamente privados, brindan algunos beneficios de privacidad cuando se usan con cuidado.

Consideraciones de enrutamiento

Como se trata en "Denegación de servicio", los nodos que abren canales públicos se exponen al riesgo de una serie de ataques a sus canales. Si bien se están desarrollando mitigaciones a nivel de protocolo, hay muchos pasos que un nodo puede tomar para protegerse contra ataques de denegación de servicio en sus canales públicos:

Tamaño mínimo de HTLC

En el canal abierto, su nodo puede establecer el tamaño mínimo de HTLC que aceptará. Establecer un valor más alto garantiza que cada uno de los espacios de canal disponibles no pueda ser ocupado por un pago muy pequeño.

Limitación de velocidad

Muchas implementaciones de nodos permiten que los nodos acepten o rechacen dinámicamente los HTLC que se reenvían a través de su nodo. Algunas pautas útiles para un limitador de velocidad personalizado son las siguientes:

- Limite la cantidad de espacios de compromiso que un solo par puede consumir
- Monitoree las tasas de fallas de un solo par y limite la tasa si sus fallas aumentan repentinamente

Canales de sombra

Los nodos que deseen abrir grandes canales a un solo objetivo pueden, en cambio, abrir un solo canal público al objetivo y admitirlo con más canales privados llamados canales ocultos. Estos canales aún se pueden usar para el enrutamiento, pero no se anuncian a los posibles atacantes.

Aceptar canales

En la actualidad, los nodos Lightning tienen dificultades para arrancar la liquidez entrante. Si bien existen algunas soluciones pagas para adquirir liquidez entrante, como servicios de intercambio, mercados de canales y servicios de apertura de canales pagos de centros conocidos, muchos nodos aceptarán con gusto cualquier solicitud de apertura de canales que parezca legítima para aumentar su liquidez entrante.

Volviendo al contexto de Bitcoin, esto se puede comparar con la forma en que Bitcoin Core trata sus conexiones entrantes y salientes de manera diferente debido a la preocupación de que el nodo pueda ser eclipsado. Si un nodo abre una conexión entrante a su nodo de Bitcoin, no tiene forma de saber si el iniciador lo seleccionó al azar o si está apuntando específicamente a su nodo con intenciones maliciosas. Sus conexiones salientes no necesitan ser tratadas con tanta sospecha porque el nodo se seleccionó al azar de un grupo de muchos pares potenciales o usted se conectó intencionalmente al par manualmente.

Lo mismo se puede decir en Lightning. Cuando abre un canal, lo hace con intención, pero cuando una parte remota abre un canal a su nodo, no tiene forma de saber si este canal se utilizará para atacar su nodo o no. Como señalan varios artículos, el costo relativamente bajo de activar un nodo y abrir canales a los objetivos es uno de los factores importantes que facilitan los ataques. Si acepta canales entrantes, es prudente colocar algunas restricciones en los nodos de los que acepta canales entrantes. Muchas implementaciones exponen ganchos de aceptación de canales que le permiten adaptar sus políticas de aceptación de canales a sus preferencias.

La cuestión de aceptar y rechazar canales es filosófica.
¿Qué pasa si terminamos con una Lightning Network donde los nuevos nodos no pueden participar porque no pueden abrir ningún canal? Nuestra sugerencia no es establecer una lista exclusiva de "mega-hubs" desde los cuales aceptará canales, sino aceptar canales de una manera que se adapte a su preferencia de riesgo.

Algunas estrategias potenciales son:

Sin riesgo

No acepte ningún canal entrante.